



Introdução à programação com Python \\ PDFDrive.com

**Bernardo Martins
Rocha
bernardomartinsrocha
@ice.ufjf.br**

VISIT...

LANZAROTE
Caliente.COM



Introdução à programação com Python \\ PDFDrive.com

**Bernardo Martins
Rocha
bernardomartinsrocha
@ice.ufjf.br**

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

Conteúdo

Parte

I

Introdução

Sintaxe

Tipos

de

dados

Construções

básicas

(if,

while,

for,

...)

Listas,

Tuplas

Funções

Parte

II

Dicionário

Orientação

a

objetos

Programação

funcional

Módulos

Arquivos

Parte

III

Computação

científica

com

Python

Parte

IV

Bibliotecas

e

programas

interessantes

2 / 180

Parte

I

Introdução

à

linguagem

Python

3 / 180



Sobre
a
linguagem
Python

Criada

por

Guido

van

Rossum

em

1991

Linguagem

de

Alto

Nível

Interpretada

Programação:

Modular

Orientada

a

objetos

Funcional

Tipagem

dinâmica

e

forte

Vasta

coleção

de

bibliotecas

Código

aberto

(GPL)

4 / 180

☐

☐

☐

☐

☐

☐

☐

☐

☐

Sobre

a

linguagem

Python

Diversas

estruturas

de

dados

nativas

lista,

tupla,

dicionário

Gerenciamento

de

memória

automático

Tratamento

de

exceções

Sobrecarga

de

operadores

Indentação

para

estrutura

de

bloco

Multiplataforma

Bem

documentada

Muito

usada

5 / 180

Google,

YouTube,

Battlefield

2

SPSS,

ESRI,

ArcGIS,

Abaqus,

OpenOffice

Blender,

GIMP,

Inkscape,

YouTube,

NASA,

CERN

Quem

usa?

2

6

6

6

6

6

6

6

6

6

6

6

Sobre

a

linguagem

Python

Diversas

estruturas

de

dados

nativas

lista,

tupla,

dicionário

Gerenciamento

de

memória

automático

Tratamento

de

exceções

Sobrecarga

de

operadores

Indentação

para

estrutura

de

bloco

Multiplataforma

Bem

documentada

Muito

usada

Quem

usa?

Blender,

GIMP,

Inkscape,

YouTube,

NASA,

CERN

SPSS,

ESRI,

ArcGIS,

Abaqus,

OpenOffice

Google,

YouTube,

Battlefield

2

6 / 180

⌂

🔍

⌂

🔍

⌂

⌂

⌂



Porque

usar

Python?

Fácil,

simples

Sintaxe

limpa

Diversas

bibliotecas

já

inclusas

Mais

expressiva

do

que

muitas

linguagens

(C/C++ ,

Perl,

Java)

Interativa

Protótipos

rápidos

Alta

produtividade

Interfaces

para

outras

linguagens

como

C/C++

e

Fortran

7 / 180



Vamos

começar

Python

é

uma

linguagem

interpretada

Não

existe

uma

etapa

de

compilação

do

código,

como

em

C/C++ ,

Fortran

Você

simplesmente

executa

o

comando

python

e

pode

começar

a

executar

código

de

forma

interativa

8 / 180

> > >

x

=

2**3

pinkfloyd

> > >

print

'pink'

+

'floyd'

4

> > >

print

2

+

2

> > >

Type

help,

copyright,

credits

or

license

for

more

info..

[GCC

4.4.5]

on

linux2

Python

2.6.6

(r266:84292,

Sep

15

2010,

15:52:39)

rocha@machine:~/Desktop/\$

python



Vamos

começar

Python

é

uma

linguagem

interpretada

Não

existe

uma

etapa

de

compilação

do

código,

como

em

C/C++ ,

Fortran

Você

simplesmente

executa

o

comando

python

e

pode

começar

a

executar

código

de

forma

interativa

rocha@machine:~/Desktop/\$

python

Python

2.6.6

(r266:84292,

Sep

15

2010,

15:52:39)

[GCC

4.4.5]

on

linux2

Type

help,

copyright,

credits

or

license

for

more

info..

> > >

> > >

print

2

+

2

4

> > >

print

'pink'

+

'floyd'

pinkfloyd

> > >

x

=

2**3

9 / 180

❏

❏

Alguns

detalhes

Não

é

preciso

terminar

comandos

com

;

Não

é

preciso

declarar

o

tipo

de

dado

das

variáveis

> > >

a

=

2**3

> > >

a

8

> > >

x

=

2.9

+

6.5

+

1.1

> > >

x

10.5

> > >

print

type(a)

< **type**

'int'>

> > >

print

type(x)

< **type**

'float'>

10 / 180

./programa.py

python

programa.py

ou

através

da

linha

de

comando

(dois

cliques):

python,

como

no

exemplo

acima

de

forma

interativa

usando

o

interpretador

Podemos

executar

códigos:



❏

❏

❏

❏

❏

Alguns

detalhes

Não

é

preciso

terminar

comandos

com

;

Não

é

preciso

declarar

o

tipo

de

dado

das

variáveis

> > >

a

=

2**3

> > >

a

8

> > >

x

=

2.9

+

6.5

+

1.1

> > >

x

10.5

> > >

print

type(a)

< **type**

'int'>

> > >

print

type(x)

< **type**

'float'>

Podemos

executar

códigos:

de

forma

interativa

usando

o

interpretador

python,

como

no

exemplo

acima

ou

através

da

linha

de

comando

(dois

cliques):

python

programa.py

./programa.py

11 / 180



9

Tipos

de

Dados

> > >

x

=

2

**

3

> > >

x

/

2

4

> > >

type(x)

int

> > >

x

=

10.5

Tipos

de

datos

básicos

> > >

type(x)

int,

long,

float,

float

complex

> > >

m

=

(6.8

+

9.4

+

8.4)/3

> > >

m

bool

8.2000000000000001

O

tipo

de

uma

variável

> > >

m

>

6.0

True

muda

conforme

o

valor

> > >

(m

> =

9.0)

and

(m

< =

10.0)

atribuído

False

> > >

c1

=

3

+

1j

> > >

c2

=

complex(-3,2)

> > >

c1

+

c2

3j

12 / 180

Tipagem

forte

> > >

c

=

"5"

> > >

q

=

4

> > >

c,

q

=

q,

c

> > >

print

c,

q

4

5

> > >

print

c

+

q

Traceback

(most

recent

call

last):

File

"<stdin>", line

1,

in

<module>

TypeError:

cannot

concatenate

'str'

and

'int'

objects

13 / 180

como

Perl

que

é

dinâmica

e

com

tipagem

frac.

tipagem

forte,

ao

contrário

de

outras

linguagens

Isto

é,

Python

é

uma

linguagem

dinâmica,

mas

com

Tipagem

forte

> > >

c

=

"5"

> > >

q

=

4

> > >

c,

q

=

q,

c

> > >

print

c,

q

4

5

> > >

print

c

+

q

Traceback

(most

recent

call

last):

File

"<stdin>", line

1,

in

<module>

TypeError:

cannot

concatenate

'str'

and

'int'

objects

Isto

é,

Python

é

uma

linguagem

dinâmica,

mas

com

tipagem

forte,

ao

contrário

de

outras

linguagens

como

Perl

que

é

dinâmica

e

com

tipagem

fraca.

14 / 180

❏

❏

Strings

Python

define

um

tipo

de

dados

nativo

para

strings

(str)

Strings

podem

ser

delimitadas

por

aspas

simples,

dupla

ou

tripla

15 / 180

'tripla

possui

uma

propriedade

especial:

elas

\n

ignoram

,

ela

eh

escrita""

ignoram

quebra

de

linha,

portanto

a

string

aparece

como

> > >

""tripla possui

uma

propriedade

especial:

elas

'tripla'

> > >

""tripla""

'dupla'

> > >

"dupla"

'simples'

> > >

'simples'



❶

❷

Strings

Python

define

um

tipo

de

dados

nativo

para

strings

(str)

Strings

podem

ser

delimitadas

por

aspas

simples,

dupla

ou

tripla

> > >

'simples'

'simples'

> > >

"dupla"

'dupla'

> > >

""*tripla*""

'tripla'

> > >

""*tripla possui*

uma

propriedade

especial:

elas

ignoram

quebra

de

linha,

portanto

a

string

aparece

como

ela

eh

escrita""

'tripla

possui

uma

propriedade

especial:

elas

\n

ignoram

,

16 / 180



Strings

> > >

```
print("C:\diretorio\novo\nada.exe") C:\diretorio
```

ovo

ada.exe

Como

evitar

a

quebra

de

linha

?

17 / 180

> > >

```
print(u"\u2192") C:\diretorio\novo\nada.exe
```

> > >

```
print(r'C:\diretorio\novo\nada.exe') unicode
```

string

raw

string

Modos

especiais

de

strings:

> > >

print('C:\diretorio\\novo\\nada.exe')



▣

Strings

> > >

print("C:\diretorio\novo\nada.exe") C:\diretorio

ovo

ada.exe

Como

evitar

a

quebra

de

linha

?

> > >

print('C:\diretorio\\novo\\nada.exe') 18 / 180

> > >

print(u"\u2192") C:\diretorio\novo\nada.exe

> > >

print(r'C:\diretorio\novo\nada.exe') unicode

string

raw

string

Modos

especiais

de

strings:



☐

⊗

⊛

⊛

Strings

> > >

print("C:\diretorio\novo\nada.exe") C:\diretorio

ovo

ada.exe

Como

evitar

a

quebra

de

linha

?

> > >

print('C:\diretorio\\novo\\nada.exe') Modos

especiais

de

strings:

raw

string

unicode

string

> > >

print(r'C:\diretorio\novo\nada.exe') C:\diretorio\novo\nada.exe

> > >

print(u"u2192") 19 / 180

☺

Strings

Strings

são

imutáveis

> > >

"hello"

+

"world"

#

concatenacao

'helloworld'

> > >

s

=

'hello'

> > >

s[0]

=

','

Traceback

(most

recent

call

last):

File

"<stdin>", line

1,

in

< module >

TypeError:

'str'

object

does

not

support

item

assignment

> > >

sn

=

’j’

+

s[1:]

> > >

sn

’jello’

20 / 180



Strings

O

operador

'+'

não

converte

automaticamente

números

ou

outros

tipos

em

strings.

A

função

str()

converte

valores

para

sua

representação

em

string.

> > >

pi

=

3.14

> > >

text

=

'o

valor

de

pi

eh

=

,

+

pi

Traceback

(most

recent

call

last):

File

"<stdin>", line

1,

in

<module>

TypeError:

cannot

concatenate

'str'

and

'float'

objects

> > >

text

=

'o

valor

de

pi

eh

=

,

+

str(pi)

21 / 180

> > >

text

=

"float = %f

inteiro = %d

string = %s"

%

(a,i,s)

> > >

s

=

'oi'

> > >

i

=

100

> > >

a

=

10.2

> > >

print

text

> > >

text

=

'o

valor

de

pi

eh

=

%f'

%

pi

Formatadores

especiais



❏

❏

Strings

O

operador

'+',

não

converte

automaticamente

números

ou

outros

tipos

em

strings.

A

função

str()

converte

valores

para

sua

representação

em

string.

> > >

pi

=

3.14

> > >

text

=

'o

valor

de

pi

eh

=

,

+

pi

Traceback

(most

recent

call

last):

File

"<stdin>", line

1,

in

<module>

TypeError:

cannot

concatenate

'str'

and

'float'

objects

> > >

text

=

'o

valor

de

pi

eh

=

,

+

str(pi)

Formatadores

especiais

> > >

text

=

'o

valor

de

pi

eh

=

%f'

%

pi

> > >

print

text

> > >

a

=

10.2

> > >

i

=

100

> > >

s

=

'oi'

> > >

text

=

"float = %f

inteiro = %d

string = %s"

%

(a,i,s)

22 / 180

0

0

Hello

0 1 2 3 4

-5 -4 -3 -2 -1

Strings

Acesso

sequencial,

em

fatias

ou

direto

(índice)

Slice:

s[start:end]

> > >

s

=

"hello"

> > >

s[0]

'h'

> > >

s[1:]

'ello'

> > >

s[1:4]

'ell'

> > >

s[:]

'hello'

> > >

s[1:100]

'ello'

23 / 180



Strings

-

Métodos

Strings

possuem

uma

grande

variedade

de

métodos:

lower,

upper,

capitalize

split,

strip,

join

find,

replace

startswith,

islower,

...

24 / 180

'python.

is.

powerful!!!'

> > >

".

".join(a)

+

"!!!"

'python.

is.

powerful'

> > >

".

".join(a)

> > >

a

=

s.lower().split()

['python',

'is',

'powerful']

> > >

s.lower().split()

'python

is

powerful'

> > >

s.lower()

> > >

s

=

"PYTHON

IS

POWERFUL"

Passo

a

passo

'python.

is.

powerful!!!s'

> > >

".

".join("PYTHON

IS

POWERFUL".lower().split())

+

"!!!"



Strings

-

Métodos

Strings

possuem

uma

grande

variedade

de

métodos:

lower,

upper,

capitalize

split,

strip,

join

find,

replace

startswith,

islower,

...

> > >

".

".join("PYTHON

IS

POWERFUL".lower().split())

+

"!!!"

'python.

is.

powerful!!!s'

25 / 180

'python.

is.

powerful!!!'

> > >

".

".join(a)

+

"!!!"

'python.

is.

powerful'

> > >

".

".join(a)

> > >

a

=

s.lower().split()

['python',

'is',

'powerful']

> > >

s.lower().split()

'python

is

powerful'

> > >

s.lower()

> > >

s

=

"PYTHON

IS

POWERFUL"

Passo

a

passo

❏

❏

❏

❏

❏

❏

Strings

-

Métodos

Strings

possuem

uma

grande

variedade

de

métodos:

lower,

upper,

capitalize

split,

strip,

join

find,

replace

startswith,

islower,

...

> > >

".

".join("PYTHON

IS

POWERFUL".lower().split())

+

"!!!"

'python.

is.

powerful!!!s'

Passo

a

passo

> > >

s

=

"PYTHON

IS

POWERFUL"

> > >

s.lower()

'python

is

powerful'

> > >

s.lower().split()

['python',

'is',

'powerful']

> > >

a

=

s.lower().split()

> > >

".

".join(a)

'python.

is.

powerful'

> > >

".

".join(a)

+

"!!!"

'python.

is.

powerful!!!'

26 / 180



☐

Strings

-

Métodos

#

split()

> > >

s

=

'monty

python

and

the

flying

circus'

> > >

print

s.split()

['monty',

'python',

'and',

'the',

'flying',

'circus']

#

opa,

uma

lista!

#

count()

> > >

print

s.count("th")

#

join()

> > >

s

=

"em

busca

do

calice

sagrado"

> > >

s2

=

s.split()

> > >

print

"/".join(s2) em/busca/do/calice/sagrado Ainda

é

possível

realizar

diversas

outras

operações

com

strings

27 / 180

0

0

0

0

0

0

0

0

0

Listas

e

Tuplas

Estruturas

de

dados

nativas:

list,

tuple

Coleções

de

objetos

heterogêneos

Crescem

até

o

limite

da

memória

Acesso

sequencial,

em

fatias

ou

direto

Métodos

para

adicionar,

remover,

ordenar,

procurar,

contar

Listas

são

mutáveis

e

tuplas

são

imutáveis

Tuplas

não

podem

ser

alteradas

depois

de

criadas

Listas

são

delimitadas

por

[

e

]

Tuplas

são

delimitadas

por

(

e

)

Tuplas

Uma

tupla

é

uma

coleção

de

objetos

separados

por

vírgula

> > >

primos

=

(2,3,5,7)

> > >

print

primos[0],

primos[-1]

2

7

> > >

t_vazia

=

()

> > >

print

len(t_vazia)

0

> > >

u_tupla

=

('oi',)

> > >

print

len(u_tupla)

1

Para

uma

tupla

com

1

elemento

apenas

é

preciso

usar

(val,)

29 / 180



❏

❏

❏

Tuplas

Pode

ter

ou

não

parênteses

para

delimitar

a

tupla

Tupla

aninhada

Heterogênea

> > >

t

=

12345,

54321,

'hello!'

#

ou

> > >

t

=

(12345,54321,'hello')

> > >

t[0]

12345

> > >

t

(12345,

54321,

'hello!')

> > >

u

=

t,

(1,

2,

3,

4,

5)

#

tuplas

podem

ser

aninhadas

> > >

u

((12345,

54321,

'hello!'),

(1,

2,

3,

4,

5))

> > >

x,y,z

=

t

#

desempacotar

tupla

> > >

print

y

54321

30 / 180



☰

Listas

"Arrays

flexíveis"

> > >

a

=

['spam',

'eggs',

100,

1234]

> > >

a

['spam',

'eggs',

100,

1234]

> > >

a[0]

'spam'

> > >

a[3]

1234

> > >

a[-2]

100

> > >

a[1:-1]

['eggs',

100]

> > >

a[:2]

+

['bacon',

2*2]

['spam',

'eggs',

'bacon',

4]

> > >

3*a[:3] + ['Boo!']

['spam',

'eggs',

100,

'spam',

'eggs',

100,

'spam',

'eggs',

100,

'Boo!']

31 / 180

⌂

⌂

⌂

⌂

⌂

Lista

-

Métodos

> > >

a

=

range(5)

> > >

print

a

Saída

> > >

a.append(5)

> > >

print

a

[0,

1,

2,

3,

4]

> > >

a.insert(0,42)

[0,

```
1,  
2,  
3,  
4,  
5]  
  
> > >
```

```
print
```

```
a  
[42,  
0,  
1,  
2,  
3,  
4,  
5]
```

```
> > >
```

```
a.reverse()
```

```
[5,  
4,  
3,  
2,  
1,  
0,  
42]
```

> > >

print

a

[0,

1,

2,

3,

4,

5,

42]

> > >

a.sort()

> > >

print

a

Outros

métodos

extend(L):

append

de

uma

lista

L

remove(x):

remove

primeira

ocorrência

de

x

index(x):

retorna

o

índice

da

primeira

ocorrência

de

x

na

lista

count(x):

retorna

o

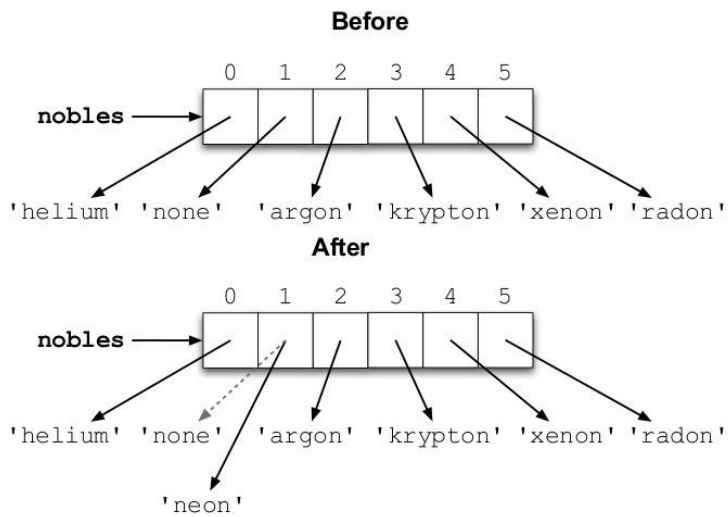
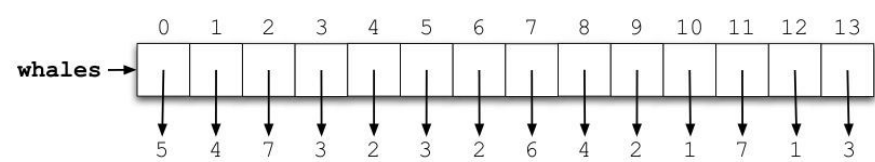
número

de

ocorrências

de

x



Lista

Representação

As

listas

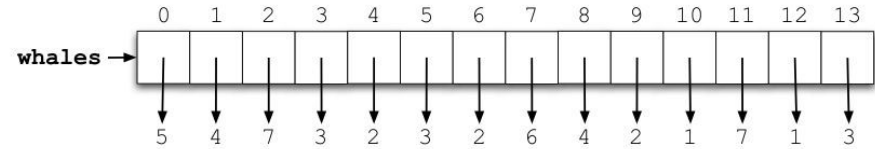
contém
ponteiros
para
objetos
que
residem
em
algum
lugar
na
memória.

33 / 180

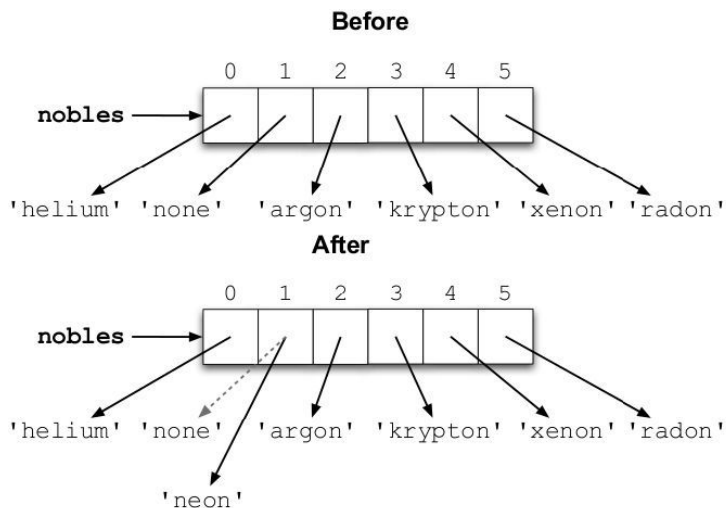
Alterações



0



0



Lista

Representação

As

listas

contém

ponteiros

para

objetos

que

residem

em

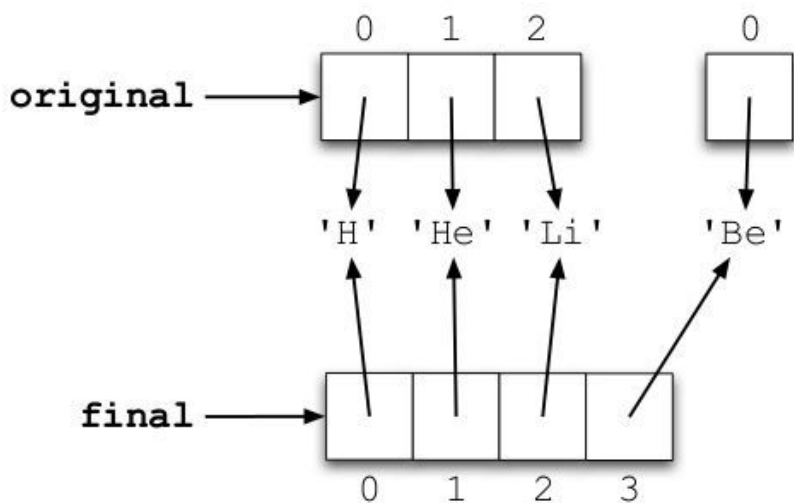
algum

lugar

na

memória.

0



Lista

Concatenação

> > >

original

=

['H',

'He',

'Li']

> > >

temp

=

['Be']

> > >

final

=

original

+

temp

> > >

print

final

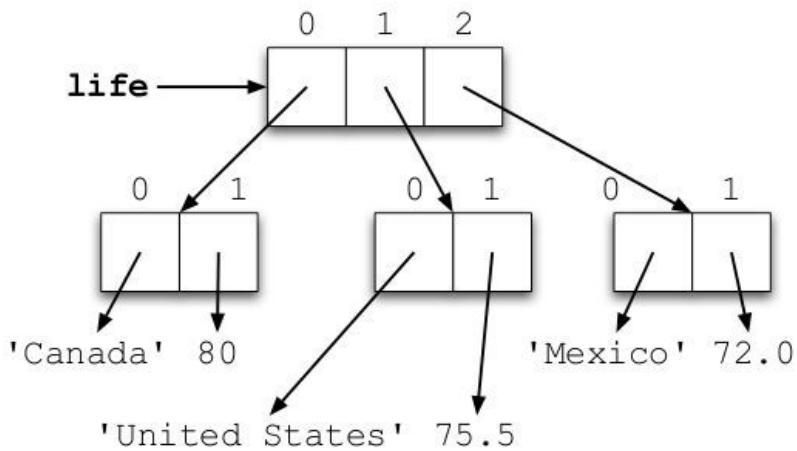
['H',

'He',

'Li',

'Be']

35 / 180



Lista

Slice

>>>

l

=

range(10)

>>>

l[2:6]

[2,

3,

4,

5]

36 / 180

4

>>>

print

```
la[2][1]
```

```
[1,
```

```
2]
```

```
> > >
```

```
print
```

```
la[1]
```

```
['a',
```

```
'b']
```

```
> > >
```

```
print
```

```
la[0]
```

```
['a',
```

```
'b'],
```

```
[1,
```

```
2],
```

```
[3,
```

```
4]]
```

```
> > >
```

```
print
```

```
la
```

```
> > >
```

```
la
```

```
=
```

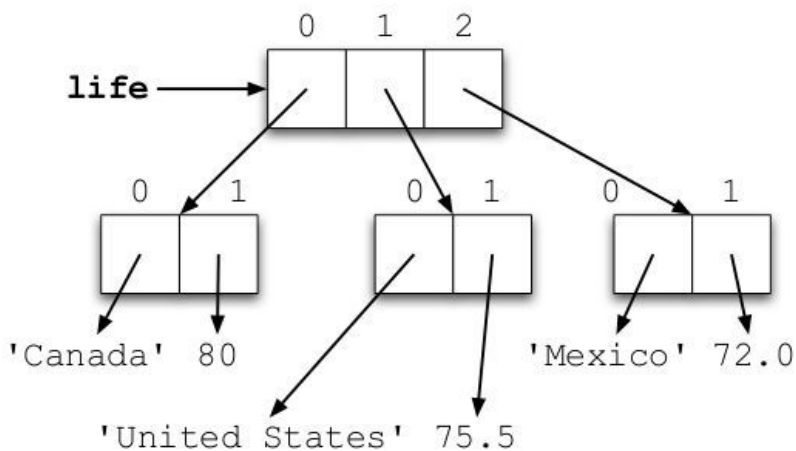
```
 [['a','b'],[1,2],[3,4]]
```

Lista

aninhada

0

0



Lista

Slice

>>>

l

=

range(10)

>>>

l[2:6]

[2,

3,

4,

5]

Lista

aninhada

37 / 180

4

> > >

print

la[2][1]

[1,

2]

> > >

print

la[1]

['a',

'b']

> > >

print

la[0]

['a',

'b'],

[1,

2],

[3,

4]]

>>>

print

la

>>>

la

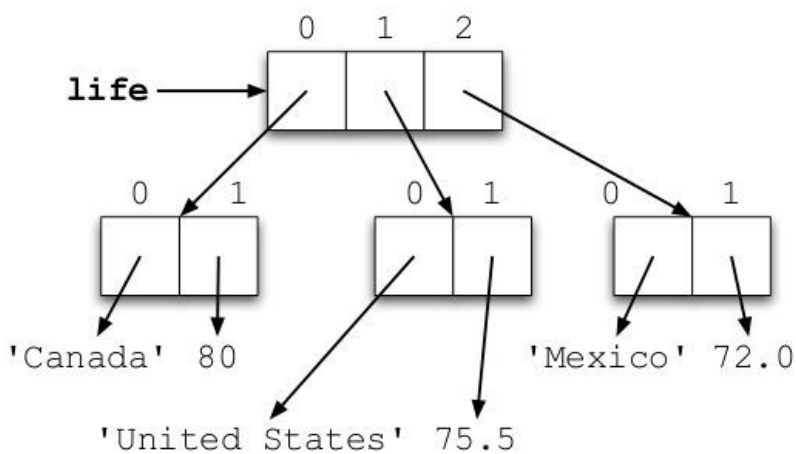
=

['a','b'],[1,2],[3,4]]



o

o



Lista

Slice

> > >

l

=

range(10)

> > >

l[2:6]

[2,

3,

4,

5]

Lista

aninhada

> > >

la

=

['a','b'],[1,2],[3,4]]

> > >

print

la

['a',

'b'],

[1,

2],

[3,

4]]

> > >

print

la[0]

['a',

'b']

> > >

print

la[1]

[1,

2]

> > >

print

la[2][1]

4

38 / 180

🔊

🔊

Built-ins

A

linguagem

Python

automaticamente

importa

algumas

funções,

tipos

e

símbolos

que

são

conhecidos

como

built-ins.

Mecanismos

básicos

da

linguagem.

39 / 180

help,

dir

listas

min,

max

e

sum

para

operações

matemáticas

em

int,

str

e

float

que

servem

para

construir

tipos

sorted

e

reversed

para

ordenar

listas

dict,

list,

set

e

object

listas

all

e

any

para

aplicar

filtros

booleanos

em

range

para

listas

Alguns

exemplos

❏

❏

❏

❏

❏

❏

Built-ins

A

linguagem

Python

automaticamente

importa

algumas

funções,

tipos

e

símbolos

que

são

conhecidos

como

built-ins.

Mecanismos

básicos

da

linguagem.

Alguns

exemplos

range

para

listas

all

e

any

para

aplicar

filtros

booleanos

em

listas

dict,

list,

set

e

object

sorted

e

reversed

para

ordenar

listas

int,

str

e

float

que

servem

para

construir

tipos

min,

max

e

sum

para

operações

matemáticas

em

listas

help,

dir

40 / 180

2. Built-in Functions

The Python interpreter has a number of functions built into it that are always available. They are listed here

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	apply()
delattr()	help()	next()	setattr()	buffer()
dict()	hex()	object()	slice()	coerce()
dir()	id()	oct()	sorted()	intern()

❏

❏

❏

Built-ins

Documentação

<http://docs.python.org/library/functions.html> <http://docs.python.org/>

41 / 180

A

função

range

> > >

help(range) 42 / 180

[20,

19,

18,

17,

16,

15,

14,

13,

12,

11]

> > >

range(20,10,-1)

[10,

12,

14,

16,

18]

> > >

range(10,20,2)

[10,

11,

12,

13,

14,

15,

16,

17,

18,

19]

> > >

range(10,20)

[0,

1,

2,

3,

4,

5,

6,

7,

8,

9]

> > >

range(10)

These

are

exactly

the

valid

indices

for

a

list

of

4

elements.

For

example,

`range(4)`

returns

`[0,`

`1,`

`2,`

`3]`.

The

end

point

is

omitted!

When

step

is

given,

it
specifies
the
increment
(or
decrement).
range(i,
j)
returns
[i,
i + 1,
i + 2,
...,
j-1];
start
(!)
defaults
to
0.
Return
a
list
containing
an

arithmetic
progression
of
integers.
range([start,
stop[,
step])

->

list
of
integers
range(...)

Help

on

built-in

function

range

in

module

`_builtin_`:



A

função

range

> > >

help(range) Help

on

built-in

function

range

in

module

`_builtin_:`

`range(...)`

`range([start,]`

`stop[,`

`step])`

->

list

of

integers

Return

a

list

containing

an

arithmetic
progression
of
integers.
`range(i,`
`j)`
returns
`[i,`
`i + 1,`
`i + 2,`
`...,`
`j-1]`;
start
(!)
defaults
to
0.
When
step
is
given,
it
specifies
the

increment

(or

decrement).

For

example,

`range(4)`

returns

`[0,`

`1,`

`2,`

`3]`.

The

end

point

is

omitted!

These

are

exactly

the

valid

indices

for

a

list

of

4

elements.

43 / 180

[20,

19,

18,

17,

16,

15,

14,

13,

12,

11]

> > >

range(20,10,-1)

[10,

12,

14,

16,

18]

> > >

range(10,20,2)

[10,

11,

12,

13,

14,

15,

16,

17,

18,

19]

> > >

range(10,20)

[0,

1,

2,

3,

4,

5,

6,

7,

8,

9]

> > >

range(10)

A

função

range

> > >

help(range) Help

on

built-in

function

range

in

module

`_builtin_:`

`range(...)`

`range([start,]`

`stop[,`

`step])`

->

list

of

integers

Return

a
list
containing
an
arithmetic
progression
of
integers.
range(i,
j)
returns
[i,
 $i + 1$,
 $i + 2$,
...,
 $j - 1$];
start
(!)
defaults
to
0.
When
step
is

given,
it
specifies
the
increment
(or
decrement).

For
example,
`range(4)`
returns
[0,
1,
2,
3].

The
end
point
is
omitted!

These
are
exactly
the

valid

indices

for

a

list

of

4

elements.

> > >

range(10)

[0,

1,

2,

3,

4,

5,

6,

7,

8,

9]

> > >

range(10,20)

[10,

11,

12,

13,

14,

15,

16,

17,

18,

19]

> > >

range(10,20,2)

[10,

12,

14,

16,

18]

> > >

range(20,10,-1)

[20,

19,

18,

17,

16,

15,

14,

13,

12,

11]

44 / 180

❏

❏

❏

bool

Tipo

de

dados

bool

True,

False

e

None

Operadores:

is,

not,

and,

or

> > >

x

=

11

> > >

x

>

0

True

> > >

x

%

2

= =

0

False

> > >

y

=

True

> > >

not

y

False

> > >

x

is

not

None

True

45 / 180

❏

❏

❏

Indentação

Aviso

aos

navegantes!

Python

não

usa

{,

},

begin,

end

para

estrutura

de

bloco

Código

em

C

if

(num

%

2

= =

0)

{

if

(num

%

2

= =

0)

{

par

=

par

+

1;

par

=

par

+

1;

printf("Par"); printf("Par");

}

}

else

{

else

impar

=

impar

+

1;

{

printf("Impar"); impar

=

impar

+

1;

```
}  
  
printf("Impar");  
  
}
```

46 / 180

indentação

do

código!

Em

Python

a

estrutura

de

bloco

é

definida

pela

❏

❏

❏

❏

Indentação

Aviso

aos

navegantes!

Python

não

usa

{,

},

begin,

end

para

estrutura

de

bloco

Código

em

C

if

(num

%

2

= =

0)

{

```
if
(
    num
    %
    2
    ==
    0
)
{
    par
    =
    par
    +
    1;
    par
    =
    par
    +
    1;
    printf("Par"); printf("Par");
}
}
else
{
    else
    impar
```

```
=  
  
impar  
  
+  
  
1;  
  
{  
  
printf("Impar"); impar  
  
=  
  
impar  
  
+  
  
1;  
  
}  
  
printf("Impar");  
  
}
```

Em

Python

a

estrutura

de

bloco

é

definida

pela

indentação

do

código!

47 / 180



Indentação

Isso

mesmo!

Tudo

que

pertence

a

um

mesmo

bloco

fica

alinhado

no

mesmo

nível

do

código

fonte.

if

num

%

2

= =

0:

par

=

par

+

1

print

'Par'

else:

impar

=

impar

+

1

print

'Impar'

48 / 180

branco

ou

2,

o

importante

é

ser

consistente.

Em

geral

programadores

Python

usam

4

espaços

em

IndentationError: expected

an

indented

block

^

print

'par'

File

"<stdin>", line

2

print

'par'

if

x

%

2

= =

0:

Erro

de

indentação



❏

❏

❏

❏

Indentação

Isso

mesmo!

Tudo

que

pertence

a

um

mesmo

bloco

fica

alinhado

no

mesmo

nível

do

código

fonte.

if

num

%

2

= =

0:

par

=

par

+

1

print

'Par'

else:

impar

=

impar

+

1

print

'Impar'

Erro

de

indentação

if

x

%

2

= =

0:

print

'par'

File

"<stdin>", line

2

print

'par'

^

IndentationError: expected

an

indented

block

Em

geral

programadores

Python

usam

4

espaços

em

branco

ou

2,

o

importante

é

ser

consistente.

Estruturas

de

Controle

If-elif-else

If-Else

if

exp:

comandos

else:

comandos

If-Else-If-Else

if

exp:

comandos

elif

exp:

comandos

else:

comandos

50 / 180

...

print

"OK"

...

else:

...

print

"Fora

do

intervalo"

> > >

if

((x < 0)

or

(x > 100)):

...

print

"Fora

do

intervalo"

...

else:

...

print

"OK"

> > >

if

((x

> =

0)

and

(x

< =

100)):

...

print

'Positivo'

...

else:

...

print

'Zero'

...

elif

x

= =

0:

...

print

'Negativo'

> > >

if

x

<

0:

> > >

x

=

int(raw_input("Numero: ")) Exemplo

Estruturas

de

Controle

If-elif-else

Exemplo

If-Else

> > >

x

=

int(raw_input("Numero:

"))

if

exp:

> > >

if

x

<

0:

comandos

...

print

'Negativo'

else:

...

elif

x

= =

0:

comandos

...

print

'Zero'

...

else:

...

print

'Positivo'

> > >

if

((x

> =

0)

and

(x

< =

100)):

If-Else-If-Else

...

print

"OK"

...

else:

if

exp:

...

print

"Fora

do

intervalo"

comandos

elif

exp:

> > >

if

((x < 0)

or

(x > 100)):

comandos

...

print

"Fora

do

intervalo"

else:

...

else:

comandos

...

print

"OK"

51 / 180

Estruturas

de

Controle

For,

While

For

While

lst

=

[10,20,30,'oi','tchau']

while

exp:

for

item

in

lst:

comandos

print

item

while

exp:

for

letra

in

"python":

if

exp2:

print

letra

comandos1

if

exp3:

for

k

in

range(100):

break

print

k

comandos2

52 / 180

worm

4

2

worm

spider

6

1

spider

cat

3

0

cat

...

...

...

print

x,

len(x)

...

print

i,

a[i]

> > >

for

x

in

a:

> > >

for

i

in

range(len(a)):

> > >

a

=

['cat',

'spider',

'worm']

Estruturas

de

Controle

For,

While

For

While

lst

=

[10,20,30,'oi','tchau']

while

exp:

for

item

in

lst:

comandos

print

item

while

exp:

for

letra

in

"python":

if

exp2:

print

letra

comandos1

if

exp3:

for

k

in

range(100):

break

print

k

comandos2

> > >

a

=

['cat',

'spider',

'worm']

> > >

for

x

in

a:

...

print

x,

len(x)

...

cat

3

spider

6

worm

4

53 / 180

2

worm

1

spider

0

cat

...

...

print

i,

a[i]

> > >

for

i

in

range(len(a)):

Estruturas

de

Controle

For,

While

For

While

lst

=

[10,20,30,'oi','tchau']

while

exp:

for

item

in

lst:

comandos

print

item

while

exp:

for

letra

in

"python":

if

exp2:

print

letra

comandos1

if

exp3:

for

k

in

range(100):

break

print

k

comandos2

> > >

a

=

['cat',

'spider',

'worm']

> > >

for

i

in

range(len(a)):

> > >

for

x

in

a:

...

print

i,

a[i]

...

print

x,

len(x)

...

...

0

cat

cat

3

1

spider

spider

6

2

worm

worm

4

54 / 180



Outras

construções

para

loops

enumerate()

A

função

enumerate()

cria

pares

uteis

> > >

nomes

=

['Ana',

'Maria',

'Carla']

> > >

for

par

in

enumerate(nomes):

...

print

par

(0,

'Ana')

(1,

'Maria')

(2,

'Clara')

> > >

for

i,nome

in

enumerate(nomes):

...

print

i,nome

0

Ana

1

Maria

2

Clara

55 / 180



Outras

construções

para

loops

zip()

A

função

zip()

recebe

um

par

de

sequências

como

entrada

e

cria

uma

tupla

com

os

seus

elementos

> > >

nomes

=

['Pato','Ganso','Hulk']

> > >

gols

=

[2,0,8]

> > >

for

n,

g

in

zip(nomes,gols):

...

print

'%s

fez

%d

gols'

%

(n,g)

...

Pato

fez

2

gols

Ganso

fez

0

gols

Hulk

fez

8

gols

56 / 180





Estruturas

de

Controle

Switch

Python

não

possui

uma

estrutura

do

tipo

switch,

como

C,

C + +

e

Java.

Podemos

contornar

a

situação

com

uma

cadeia

de

if-elses

> > >

if

n

= =

0:

...

print

'Voce

digitou

zero.'

...

elif

n

= =

1:

...

print

'Voce

digitou

um.'

...

elif

n

= =

2:

...

print

'Voce

digitou

dois.'

...

elif

n

= =

3:

...

print

'Voce

digitou

tres.'

...

else:

...

print

'Voce

digitou

qualquer

coisa.'

57 / 180

Funções

Procedimento

def

nome(arg1,

arg2,

...):

comandos

return

Função

def

nome1(arg1,

arg2,

...):

comandos

return

expressao

def

nome2(arg1,

arg2,

...):

comandos

return

exp1,

exp2,

exp3

def

nome3(arg1,

arg2,

argx = valor):

comando

return

exp

58 / 180

> > >

fib(8)

#

0

1

1

2

3

5

> > >

par(6)

#

True

...

a,

b

=

b,

a + b

...

print

a,

...

while

a

<

n:

...

a,

b

=

0,

1

...

"""

Imprime

ate

n. """

> > >

def

fib(n):

...

return

(n

%

2

= =

0)

> > >

def

par(n):

Funções

Procedimento

def

nome(arg1,

arg2,

...):

comandos

> > >

def

par(n):

return

...

return

(n

%

2

= =

0)

Função

> > >

def

fib(n):

...

"""

Imprime

ate

n. ""

def

nome1(arg1,

arg2,

...):

...

a,

b

=

0,

1

comandos

...

while

a

<

n:

return

expressao

...

print

a,

...

a,

b

=

b,

a + b

def

nome2(arg1,

arg2,

...):

comandos

> > >

par(6)

#

True

return

exp1,

exp2,

exp3

> > >

fib(8)

#

0

1

1

2

3

5

def

nome3(arg1,

arg2,

argx = valor):

comando

return

exp

59 / 180

🔍

Funções

Podemos

criar

funções

com

parâmetros

opcionais

que

possuem

um

valor

default

pré-definido

> > >

def

mult(x,

num = 2):

...

return

x,

x*num

> > >

a,b

=

mult(2)

> > >

print

a,b

#

2

4

> > >

a,b

=

mult(2,
num = 10)

> > >

print

a,b

#

2

20

> > >

a,b

=

mult(3,
5)

> > >

print

a,b

#

3

15

60 / 180



Funções

Exemplo

def

divide(a,

b):

"""

Divide

operando

a

e

b

usando

divisao

inteira.

Returna

o

quociente

e

resto

da

divisao

em

uma

tupla.

''''

q

=

a

//

b

r

=

a

-

q

*

b

return

q,

r

Uso

> > >

div(10,2)

(5,

0)

> > >

mq,

mr

=

div(10,3)

> > >

print

mq,

mr

3

1

> > >

help(divide)

Help

on

function

divide

in

module

__main__:

divide(a,

b)

Divide

operando

a

e

b

usando

divisao

inteira.

Retorna

o

quociente

e

resto

da

divisao

em

uma

tupla.

61 / 180

🔗

🔗

🔗

🔗

🔗

EXERCÍCIO

Escreva

uma

função

que

dada

uma

string

que

representa

uma

URL

de

uma

página

da

web,

obter

apenas

o

endereço

da

página

principal.

Exemplo:

> > >

```
url_parse('http://www.facebook.com/fulano/photos')
```

```
'www.facebook.com'
```

Dicas:

string

split

slice

acesso

sequencial

Crie

um

arquivo

texto

para

codificar

a

sua

função.

Veja

o

exemplo

a

seguir.

62 / 180

EXERCÍCIO

def

url_parse(url):

"""

Implemente

a

funcao

abaixo

"""

pass

if

__name__ == "__main__": urlteste

=

raw_input()

print

url_parse(urlteste)

EXERCÍCIO

Uma

possível

solução

def

url_parse(url):

"""

Essa

funcao

recebe

uma

URL

valida

e

retorna

a

string

contendo

o

nome

da

pagina

principal.

Por

exemplo:

url_parse('http://semcomp.icmc.usp.br/programacao') retorna

'semcomp.icmc.usp.br'

""

proto,

resto

=

url.split(':')

resto

=

resto[2:]

temp

=

resto.split('/')

host

=

temp[0]

return

host



❏

❏

❏

EXERCÍCIO

Módulo

Vamos

supor

que

você

tenha

codificado

a

função

`url_parse()`

em

um

arquivo

fonte

chamado

`teste.py`

Como

posso

usar

essa

função

em

outros

programas?

65 / 180

semcomp.icmc.usp.br

> > >

print

`url_parse("http://semcomp.icmc.usp.br/programacao")`

> > >

from

teste

import

`url_parse`

forma

Basta

usar

os

comandos

from

e

import

da

seguinte

❏

❏

❏

❏

EXERCÍCIO

Módulo

Vamos

supor

que

você

tenha

codificado

a

função

url_parse()

em

um

arquivo

fonte

chamado

teste.py

Como

posso

usar

essa

função

em

outros

programas?

Basta

usar

os

comandos

from

e

import

da

seguinte

forma

> > >

from

teste

import

url_parse

> > >

print

url_parse("http://semcomp.icmc.usp.br/programacao")
semcomp.icmc.usp.br

66 / 180

Parte

II

Uma

introdução

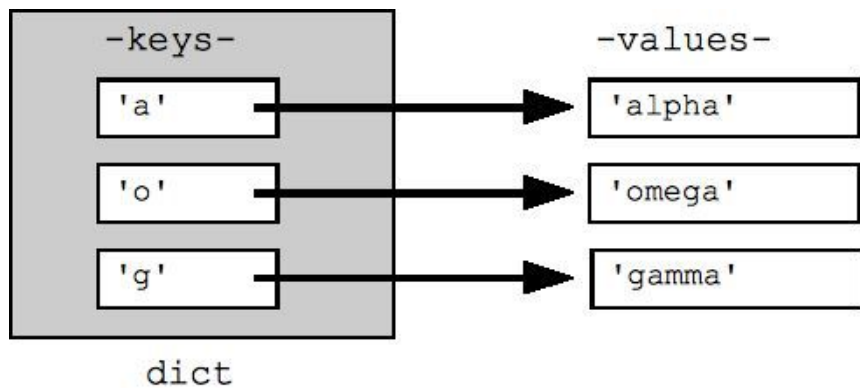
um

pouco

mais

avançada

67 / 180



Dicionário

Dicionário

é

uma

estrutura

de

dados

muito

útil

que

permite

armazenar

e

recuperar

pares

de

chaves-e-valores.

Arrays

associativos.

De

forma

grosseira

podemos

dizer

que

um

dicionário

é

uma

lista

que

podemos

acessar

seus

elementos

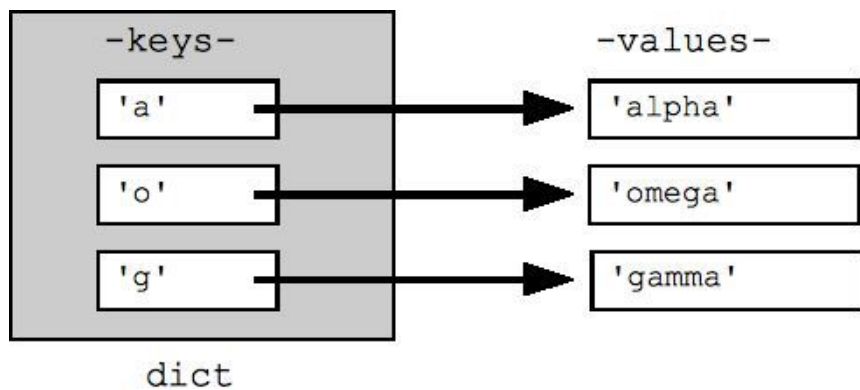
através

de

strings.

68 / 180





Dicionário

Dicionário

é

uma

estrutura

de

dados

muito

útil

que

permite

armazenar

e

recuperar

pares

de
chaves-e-valores.

Arrays

associativos.

De

forma

grosseira

podemos

dizer

que

um

dicionário

é

uma

lista

que

podemos

acessar

seus

elementos

através

de

strings.

Dicionário

> > >

d

=

{}

> > >

d['paulo']

=

25

> > >

d['jose']

=

16

> > >

d['alice']

=

21

> > >

print

d

{'paulo':

25,

'jose':

16,

'alice':

21}

> > >

print

d['alice']

21

> > >

d['alice']

=

'Paris'

> > >

print

d

{'paulo':

25,

'jose':

16,

'alice':

'Paris'}

> > >

'jose'

in

d

True

> > >

c

=

{'Juiz

de

Fora':

25.0,

'Oxford':

'rain',

'Rio

de

Janeiro':

40.0}

> > >

print

c['Petropolis']

#

KeyError

> > >

if

'Petropolis'

in

c:

print

c['Petropolis']

#

evita

KeyError

> > >

print

c.get('San

Diego')

#

retorna

None

None

> > >

novo

=

dict(a=10,

b=20,

c=30)

> > >

print

novo

{'a':

10,

'c':

30,

'b':

20}

70 / 180



Dicionário

Percorrendo

dicionários

> > >

d

=

dict(c=10,

b=20,

a=30)

> > >

for

key

in

d:

...

print

key

> > >

print

d.keys()

['a',

'c',

'b']

> > >

print

d.values()

[10,

30,

20]

#

loop

sobre

as

chaves

de

forma

ordenadas

> > >

for

key

in

sorted(d.keys()):

...

print

key,

d[key]

#

retorna

uma

lista

onde

cada

elemento

eh

uma

tupla

(chave,valor)

> > >

print

d.items()

[('a',

10),

('c',

30),

('b',

20)]

> > >

for

k,v

in

d.items():

print

k,

'->',

v

71 / 180

🔊

List

comprehension

Como

vimos

podemos

iterar

trabalhar

com

listas

da

seguinte

forma

> > >

lista

=

[2,4,6,8,10]

> > >

nova

=

[]

> > >

for

x

in

lista:

...

nova.append(x*x)

72 / 180

[4,

16,

36,

64]

> > >

print

nova

> > >

nova

=

[

x*x

for

x

in

lista

]

operação.

sintaxe

mais

compacta

para

realizar

esse

tipo

de

Entretanto

a

linguagem

Python

fornece

uma

❏

❏

List

comprehension

Como

vimos

podemos

iterar

trabalhar

com

listas

da

seguinte

forma

> > >

lista

=

[2,4,6,8,10]

> > >

nova

=

[]

> > >

for

x

in

lista:

...

nova.append(x*x)

Entretanto

a

linguagem

Python

fornece

uma

sintaxe

mais

compacta

para
realizar
esse
tipo
de
operação.

> > >

nova

=

[

x*x **for** x **in** lista]

> > >

print

nova

[4,

16,

36,

64]

73 / 180



Q

List

comprehension

Vejamos

agora

como

realizar

a

seguinte

operação

usando

list

comprehension

> > >

lista

=

[2,3,6,7,8,9,10,11]

> > >

nova

=

[]

> > >

for

x

in

lista:

...

if

$(x \% 2) == 0$:

...

`nova.append(str(x))`

...

> > >

print

`nova`

`['2',`

`'6',`

`'8',`

`'10']`

> > >

74 / 180

Muito

mais

simples

e

elegante,

não?

como

uma

espécie

de

filtro.

Essa

nova

versão

introduz

uma

expressão

que

atua

> > >

nova

=

[**str**(x)

for

x

in

lista

if(x%2 == 0)]

Podemos

reescrever

da

seguinte

forma

❏

❏

❏

❏

List

comprehension

Vejamos

agora

como

realizar

a

seguinte

operação

usando

list

comprehension

> > >

lista

=

[2,3,6,7,8,9,10,11]

> > >

nova

=

[]

> > >

for

x

in

lista:

...

if

(x%2) == 0:

...

nova.append(**str**(x))

...

> > >

print

nova

['2',

'6',

'8',

'10']

> > >

Podemos

reescrever

da

seguinte

forma

> > >

nova

=

[**str**(x)]

for

x

in

lista

if(x%2 == 0)]

Essa

nova

versão

introduz

uma

expressão

que

atua

como

uma

espécie

de

filtro.

Muito

mais

simples

e

elegante,

não?

75 / 180



☰

List

comprehension

Outro

exemplo

> > >

texto

=

"There

is

someone

in

my

head

but

it

is

not

me".split()

> > >

nova

=

[(p.upper(),

p.lower(),

len(p))

for

p

in

texto]

> > >

print

nova

[('THERE',

'there',

5),

('IS',

'is',

```
2),  
(‘SOMEONE’,  
 ‘someone’,  
7),  
...  
]
```

76 / 180

```
[‘plotPerfusionCoefs.py’,  
 ‘findSurf.py’,  
...]
```

```
> > >
```

```
files
```

```
=
```

```
[f
```

```
for
```

```
f
```

```
in
```

```
glob(‘*.py’)]
```

```
> > >
```

```
from
```

```
glob
```

```
import
```

```
glob
```

```
> > >
```

import

os

Lista

com

todos

arquivos

.py

de

um

diretório

☺

☺

List

comprehension

Outro

exemplo

> > >

texto

=

"There

is

someone

in

my

head

but

it

is

not

me".split()

> > >

nova

=

[(p.upper(),

p.lower(),

len(p))

for

p

in

texto]

> > >

print

nova

[('THERE',

'there',

5),
(‘IS’,
‘is’,
2),
(‘SOMEONE’,
‘someone’,
7),
...

]
Lista
com
todos
arquivos
.py
de
um
diretório

> > >

import

os

> > >

from

glob

import

glob

> > >

files

=

[f

for

f

in

glob('*.*py')]

['plotPerfusionCoefs.py',

'findSurf.py',

...]

77 / 180

🔗

🔗

Classes

Vamos

apresentar

de

forma

rápida

como

construir

classes

em

Python

através

de

alguns

exemplos.

A

classe

mais

simples

do

mundo

class

Ponto:

pass

> > >

p

=

Ponto()

> > >

print

p

<__main__.Ponto

instance

at

0x7f891f392098 >

78 / 180

p

=

Ponto(2.0,

1.0)

self.yCoord

=

y

self.xCoord

=

x

def

__init__(self,

x,

y):

class

Ponto:

Construtor



Classes

Vamos

apresentar

de

forma

rápida

como

construir

classes

em

Python

através

de

alguns

exemplos.

A

classe

mais

simples

do

undo

class

Ponto:

pass

> > >

p

=

Ponto()

> > >

print

p

<_main_.Ponto

instance

at

0x7f891f392098 >

Constructor

class

Ponto:

def

__init__(self, x, y): self.xCoord

=

x

self.yCoord

=

y

p

=

Ponto(2.0,

1.0)

79 / 180

⦿

⦿

Classes

self

é

um

parâmetro

especial

que

precisa

ser

incluído

na

definição

de

cada

método

e

precisa

ser

o

primeiro

parâmetro.

Quando

um

método

é

invocado,

esse

parâmetro

é

automaticamente

preenchido

com

a

referência

ao

objeto

no

qual

o

método

foi

invocado.

80 / 180

print

p.getX(),

p.getY()

p

=

Ponto(3.0,

1.5)

return

self.yCoord

def

getY(self):

return

self.xCoord

def

getX(self):

self.yCoord

=


```
y
self.xCoord
=
```

```
x
```

```
def
```

```
__init__(self,
```

```
x,
```

```
y):
```

```
class
```

```
Ponto:
```

```


```

```


```

```


```

```
Classes
```

```
self
```

```
é
```

```
um
```

```
parâmetro
```

```
especial
```

```
que
```

```
precisa
```

```
ser
```

incluído

na

definição

de

cada

método

e

precisa

ser

o

primeiro

parâmetro.

Quando

um

método

é

invocado,

esse

parâmetro

é

automaticamente

preenchido

com

a

referência

ao

objeto

no

qual

o

método

foi

invocado.

class

Ponto:

def

__init__(self, x, y): self.xCoord

=

x

self.yCoord

=

y

def

getX(self):

return

self.xCoord

def

getY(self):

return

self.yCoord

p

=

Ponto(3.0,

1.5)

print

p.getX(),

p.getY()

81 / 180

❏

Classes

Vamos

criar

um

método

para

alterar

o

estado

de

um

Ponto

class

Ponto:

#

...

def

shift(self,

xInc,

yInc):

self.xCoord

+ =

xInc

self.yCoord

+ =

yInc

82 / 180

print

"Distancia

=

",

p2.distancia(p1)

p2.shift(1.0,

1.0)

p1

=

Ponto(0,0);

p2

=

Ponto(1.0,1.0)

return

math.sqrt(dx**2

+

dy**2)

dy

=

self.yCoord

-

pt.yCoord

dx

=

self.xCoord

-

pt.xCoord

def

distancia(self,

pt):

#

...

class

Ponto:

Calcular

a

distância



Classes

Vamos

criar

um

método

para

alterar

o

estado

de

um

Ponto

class

Ponto:

#

...

def

shift(self,

xInc,

yInc):

self.xCoord

+ =

xInc

self.yCoord

+ =

yInc

Calcular

a

distância

class

Ponto:

#

...

def

distancia(self,

pt):

dx

=

self.xCoord

-

pt.xCoord

dy

=

self.yCoord

-

pt.yCoord

return

math.sqrt(dx**2 + dy**2) 83 / 180

print

"Distancia

=

",

p2.distancia(p1)

p2.shift(1.0,

1.0)

p1

=

Ponto(0,0);

p2

=

Ponto(1.0,1.0)

☐

☐

Classes

Vamos

criar

um

método

para

alterar

o

estado

de

um

Ponto

class

Ponto:

#

...

def

shift(self,

xInc,

yInc):

self.xCoord

+ =

xInc

self.yCoord

+ =

yInc

Calcular

a

distância

class

Ponto:

#

...

def

distancia(self,

pt):

dx

=

self.xCoord

-

pt.xCoord

dy

=

self.yCoord

-

pt.yCoord

return

math.sqrt(dx**2 + dy**2) p1

=

Ponto(0,0);

p2

=

Ponto(1.0,1.0)

```
p2.shift(1.0,
```

```
1.0)
```

```
print
```

```
"Distancia
```

```
=
```

```
",
```

```
p2.distancia(p1)
```

```
84 / 180
```

Classes

Usando

módulos

#

Arquivo

ponto.py

```
import
```

```
math
```

```
class
```

```
Point
```

```
:
```

```
def
```

```
__init__( self, x, y ): self.xCoord
```

=

x

self.yCoord

=

y

def

getX(

self

):

return

self.xCoord

def

getY(

self

):

return

self.yCoord

def

shift(

self,

xInc,

yInc

):

self._xCoord

+ =

xInc

self._yCoord

+ =

yInc

def

distance(

self,

otherPoint

):

xDiff

=

self.xCoord

-

otherPoint.xCoord

yDiff

=

self.yCoord

-

otherPoint.yCoord

return

math.sqrt(

xDiff

**

2

+

yDiff

**

2

)

85 / 180

🏠

Classes

Usando

módulos

Podemos

usar

a

classe

Ponto

da

seguinte

forma

from

ponto


```
import
```

```
Ponto
```

```
p1
```

```
=
```

```
Ponto(5,7)
```

```
p2
```

```
=
```

```
Ponto(0,0)
```

```
x
```

```
=
```

```
p1.getX()
```

```
y
```

```
=
```

```
p1.getY()
```

```
print(
```

```
"("
```

```
+
```

```
str(x)
```

```
+
```

```
",
```

```
"
```

```
+
```

```
str(y)
```

```
+
```

)"

)

p1.shift(4,

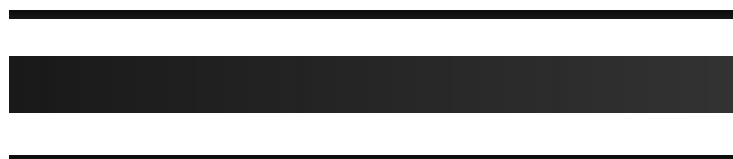
12)

d

=

p1.distancia(p2)

86 / 180



✎

✎

✎

✎

✎

Classes

Escondendo

atributos

Ao

contrário

da

maioria

das

linguagens

que

suportam

orientação

a

objetos,

Python

não

possui

um

mecanismo

para

esconder

ou

proteger

os

atributos

de

uma

classe

de

acessos

externos.

Em

C + +

temos

os

modificadores:

public,

private

e

protected

O

responsável

pela

classe

é

que

deve

indicar

quais

atributos

e

quais

métodos

devem

ser

protegidos.

E
fica
como
responsabilidade
do
usuário
da
classe,
não
violar
essa
proteção.
Ainda
assim
é
possível
"emular" esse tipo
de
proteção,
basta
acrescentar
dois
underlines
na

frente

do

nome

de

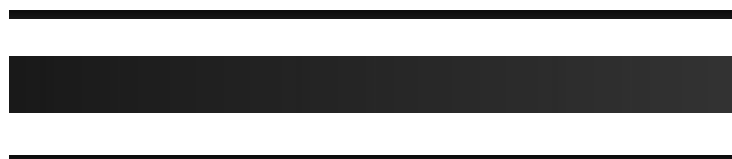
um

atributo

ou

método.

87 / 180



Classes

Escondendo

atributos

Repare

que

na

implementação

anterior

da

classe

Ponto

não

protegemos

os

atributos

xCoord

e

yCoord.

Isso

permite

que

um

usuário

altere

os

atributos

internos

da

classe

Ponto

class

Ponto:

def

```
_init_(self,x,y):
```

```
self.xCoord
```

```
=
```

```
x
```

```
self.yCoord
```

```
=
```

```
y
```

```
> > >
```

```
p
```

```
=
```

```
Ponto(2.0,2.0)
```

```
> > >
```

```
print
```

```
p.xcoord
```

```
2.0
```

```
> > >
```

```
p.xCoord
```

```
=
```

```
'zebra'
```

```
> > >
```

```
print
```

```
p.xCoord
```

```
zebra
```

```
O
```


ideal

é

que

o

usuário

só

altere

o

estado

do

objeto

através

de

métodos

que

operem

sobre

o

mesmo,

e

não

manipulando

os

seus

atributos.

88 / 180



Classes

Escondendo

atributos

Python

permite

emular

esse

ocultamento

de

informação

da

seguinte

class

Linha:

def

`_init_(self, pA, pB): self._pontoA`

=

pA

#

atributo

protegido

self._pontoB

=

pB

#

atributo

protegido

def

pontoA(self):

return

self._pontoA

def

pontoB(self):

return

self._pontoB

def

comprimento(self):

return

self._pontoA.distancia(self._pontoB) **def**

ehVertical(self):

return

self._pontoA.getX()

= =

self._pontoB.getX()

Ainda

assim

em

Python,

existem

formas

do

usuário

acessar

os

atributos

diretamente.

89 / 180

🔒

🔒

Classes

Sobrecarga

de

operadores

Em

Python

podemos

implementar

e

definir

a

funcionalidade

de

diversos

operadores

como

+,

*,

= =

como

parte

de

nossas

classes.

class

Ponto:

#

...

def

`_eq_(self, outroPonto): result`

`=`

`self.xCoord`

`= =`

`outroPonto.xCoord`

and

`\`

`self.yCoord`

`= =`

`outroPonto.yCoord`

return

`result`

Exemplo

`> > >`

`p1`

`=`

`Ponto(1.0,1.0)`

`> > >`

`p2`

`=`

`Ponto(0.0,0.0)`

`> > >`

p2.shift(1.0,1.0)

> > >

if

p1

= =

p2:

...

print

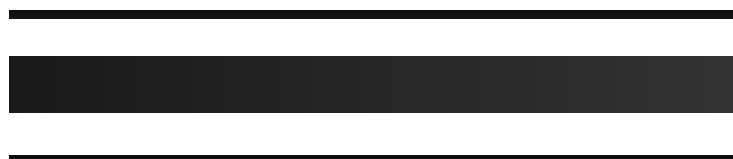
"Os

pontos

sao

iguais."

90 / 180



Classes

Sobrecarga

de

operadores

Mais

um

exemplo

de

sobrecarga

class

Ponto:

#

...

def

__str__(self):

return

"(

%f

,

%f

)"

%

(self.xCoord,

self.yCoord)

#

...

> > >

p

=

Ponto(1.5,

2.5)

> > >

print

p

(

1.500000,

1.500000

)

91 / 180



str(obj)	__str__(self)
len(obj)	__len__(self)
item in obj	__contains__(self, item)
y = obj[ndx]	__getitem__(self, ndx)
obj[ndx] = value	__setitem__(self, ndx, value)
obj == rhs	__eq__(self, rhs)
obj < rhs	__lt__(self, rhs)
obj <= rhs	__le__(self, rhs)
obj != rhs	__ne__(self, rhs)
obj > rhs	__gt__(self, rhs)
obj >= rhs	__ge__(self, rhs)

obj + rhs	__add__(self, rhs)
obj - rhs	__sub__(self, rhs)
obj * rhs	__mul__(self, rhs)
obj / rhs	__truediv__(self, rhs)
obj // rhs	__floordiv__(self, rhs)
obj % rhs	__mod__(self, rhs)
obj ** rhs	__pow__(self, rhs)
obj += rhs	__iadd__(self, rhs)
obj -= rhs	__isub__(self, rhs)
obj *= rhs	__imul__(self, rhs)
obj /= rhs	__itruediv__(self, rhs)
obj //= rhs	__ifloordiv__(self, rhs)
obj %= rhs	__imod__(self, rhs)
obj **= rhs	__ipow__(self, rhs)

Classes

Sobrecarga

de

operadores

92 / 180





0

0

0

0

0

Programação

Funcional

Além

de

suportar

programação

estruturada

e

orientação

a

objetos,

Python

também

possui

recursos

de

programação

funcional.

Vamos

apresentar

de

forma

prática

alguns

destes

mecanismos:

Funções

lambda

map,

filter

e

reduce

Existem

muitos

outros

recursos

de

programação

funcional

como

iterators

e

generators,

que

não

teremos

tempo

de

discutir.

93 / 180



⦿

⦿

Programação

Funcional

map

A

função

map

recebe

uma

sequência

(ex:

lista)

e

aplica

uma

função

a

cada

um

de

seus

elementos

e

retorna

uma

sequência

com

o

resultado

da

aplicação

da

função.

Calcular

o

quadrado

dos

elementos

de

uma

lista

> > >

def

square(num):

return

num*num

> > >

print

map(square,

range(5))

[0,

1,

4,

9,

16]

94 / 180

[10,

12,

14,

16,

18]

> > >

print

map(sum, range(5),

range(10,15))

[10,

11,

12,

13,

14]

> > >

print

range(10,15)

[0,

1,

2,

3,

4]

> > >

print

range(5)

> > >

def

sum(a,b):

return

a

+

b

Somar

elemento

a

elemento

de

duas

listas

⦿

⦿

⦿

Programação

Funcional

map

A

função

map

recebe

uma

sequência

(ex:

lista)

e

aplica

uma

função

a

cada

um

de

seus

elementos

e

retorna

uma

sequência

com

o

resultado

da

aplicação

da

função.

Calcular

o

quadrado

dos

elementos

de

uma

lista

> > >

def

square(num):

return

num*num

> > >

print

map(square,

range(5))

[0,

1,

4,

9,

16]

Somar

elemento

a

elemento

de

duas

listas

> > >

def

sum(a,b):

return

a

+

b

> > >

print

range(5)

[0,

1,

2,

3,

4]

> > >

print

range(10,15)

[10,

11,

12,

13,

14]

> > >

print

map(sum, range(5),

range(10,15))

[10,

12,

14,

16,

18]

95 / 180

🔍

Programação

Funcional

filter

A

função

filter

recebe

um

predicato

e

retorna

apenas

os

elementos

da

sequência

para

os

quais

o

predicado

resulta

no

valor

True.

> > >

def

f():

return

x

%

2

= =

0

> > >

print

filter(f,

range(5))

[0,

2,

4]

96 / 180

🔗

🔗

Programação

Funcional

reduce

A

função

reduce

começa

com

um

valor

inicial,

e

reduz

a

sequência

até

um

único

valor

aplicando

a

função

em

cada

um

dos

elementos

da

sequência

junto

com

o

valor

atual

reduzido.

Calcular

a

soma

dos

quadrados

dos

números

de

0

a

4

de

uma

lista

> > >

def

soma(reduced,num):

return

reduced

+

num*num

> > >

print

reduce(soma,

range(5),

0)

30

97 / 180

Programação

Funcional

#map

seq

=

[]

for

num

in

range(5)

:

seq

=

seq

+

[num

*

num]

print

seq

#filter

seq

=

[]

for

num

in

range(5)

:

if

num

%

2

= =

0

```
:  
  
seq  
  
=  
  
seq  
  
+  
  
[num]  
  
print  
  
seq  
  
#reduce  
  
total  
  
=  
  
0  
  
for  
  
num  
  
in  
  
range(5)  
  
:  
  
total  
  
=  
  
total  
  
+  
  
(num  
  
*  
  
num)
```

print

total

98 / 180



❏

❏

Programação

Funcional

Python

suporta

a

criação

de

funções

anônimas

(i.e:

funções

que

não

estão

ligadas

a

um

nome)

em

tempo

de

execução,

usando

uma

construção

através

da

palavra

chave

lambda.

> > >

def

f

(x):

return

$x^{**}2$

> > >

print

f(8)

64

> > >

> > >

g

=

lambda

x:

x**2

> > >

print

g(8)

64

Funções

lambda

não

precisam

de

usar

a

palavra

chave

return

99 / 180



Programação

Funcional

Vejamos

um

exemplo

mais

interessante

> > >

def

make_incrementor

(n):

return

lambda

x:

x

+

n

> > >

> > >

f

=

make_incrementor(2)

> > >

g

=

make_incrementor(6)

> > >

> > >

print

f(42),

g(42)

44

48

100 / 180

30

> > >

print

reduce(lambda r,

n:

r

+

n*n,

range(5),0)

[0,2,4]

> > >

print

filter(lambda x:

x

%

2

= =

0,

range(5))

[0,

1,

4,

9,

16]

> > >

print

map(lambda

x:

x**2,

range(5))

é

muito

prático

O

uso

de

funções

lambda

com

map,

filter

e

reduce

☺

☺

Programação

Funcional

Vejamos

um

exemplo

mais

interessante

> > >

def

make_incrementor

(n):

return

lambda

x:

x

+

n

> > >

> > >

f

=

make_incrementor(2)

> > >

g

=

make_incrementor(6)

> > >

> > >

print

f(42),

g(42)

44

48

O

uso

de

funções

lambda

com

map,

filter

e

reduce

é

muito

prático

> > >

print

map(lambda

x:

x2, range(5))**

[0,

1,

4,

9,

16]

> > >

print

filter(lambda x:

x

%

2

= =

0,

range(5))

[0,2,4]

> > >

print

reduce(lambda r,

n:

r

+

n*n, range(5),0) 30

101 / 180



Arquivos

Arquivos

são

um

tipo

built-in

do

Python

que

são

representados

por

objetos.

Ou

seja,

não

é

preciso

de

importar

módulos

para

trabalhar

com

arquivos

em

seu

programa.

Antes

de

um

arquivo

ser

usado

é

preciso

primeiro

criar

um

objeto

que

representa

o

arquivo

e

então

abrir

o

mesmo.

infile

=

```
open('dados.txt',
```

```
'r')
```

```
outfile
```

```
=
```

```
open('notas.txt',
```

```
'w')
```

Depois

que

o

processamento

com

os

arquivos

termina,

é

preciso

fechar

os

mesmos

```
infile.close()
```

```
outfile.close()
```

102 / 180



Arquivos

Escrevendo

em

arquivos

outfile

=

open('notas.txt','w') outfile.write('Notas

da

Prova\n')

outfile.write('-'

*

40

+

'\n')

for

e

in

estudantes:

outfile.write('%s

\t

%6.2f

\n'

%

(e.nome,

e.nota))

outfile.write('-'

*

40

+

'\n')

outfile.close

É

preciso

explicitamente

colocar

para

quebra

de

linha

no

comando

write,

diferentemente

de

print

Arquivos

Lendo

de

arquivos

infile

=

open(

"data.txt",

"r"

)

line

=

infile.readline()

while

line

!=

""

:

print(

line

)

line

=

infile.readline()

infile.close()

104 / 180

nome,

idade,

nota

=

temp[0],

temp[1],

temp[2]

temp

=

line.split()

line

=

infile.readline()

#

nome

idade

nota

#

linhas

no

formato

Ou

podemos

quebrar

a

linha

em

várias

partes

sline

=

line.rstrip()

line

=

infile.readline()

espacos

em

branco

à

direita

Podemos

usar

o

metodo

rstrip()

para

remover

❶

❷

❸

Arquivos

Lendo

de

arquivos

infile

=

open(

"data.txt",

"r"

)

line

=

```
infile.readline()
```

```
while
```

```
line
```

```
!=
```

```
"""
```

```
:
```

```
print(
```

```
line
```

```
)
```

```
line
```

```
=
```

```
infile.readline()
```

```
infile.close()
```

Podemos

usar

o

metodo

`rstrip()`

para

remover

espacos

em

branco

à

direita

line

=

infile.readline()

sline

=

line.rstrip()

Ou

podemos

quebrar

a

linha

em

várias

partes

#

linhas

no

formato

#

nome

idade

nota

line

=

infile.readline()

temp

=

line.split()

nome,

idade,

nota

=

temp[0],

temp[1],

temp[2]

105 / 180

❏

❏

❏

EXERCÍCIO

Vamos

implementar

uma

função

que
recebe
a
string
com
o
nome
de
um
arquivo
texto,
contendo
as
coordenadas
de
um
conjunto
de
pontos
2D,
lê
o
seu
conteúdo

e

retorna

uma

listas

com

objetos

Ponto.

Teremos

que

usar

a

classe

Ponto

definida

anteriormente.

Exemplo:

> > >

arquivo

=

"pontos.txt"

> > >

pts

=

le_pontos(arquivo)

> > >

print

pts

[<_main_.Ponto

instance

at

0x0538F3C8>,

<_main_.Ponto

instance

at

0x0538F440>,

<_main_.Ponto

instance

at

0x0538F260>,

<_main_.Ponto

instance

at

0x0538F3F0>]

106 / 180

EXERCÍCIO

Considere

arquivos

de

entrada

do

tipo

9

0.0

0.0

1.0

0.0

2.0

0.0

0.0

1.0

1.0

1.0

2.0

1.0

0.0

2.0

1.0

2.0

2.0

2.0

107 / 180



🔊

EXERCÍCIO

Uma

possível

solução

def

le_pontos(arquivo):

l

=

[]

f

=

open(arquivo,

'r')

line

=

f.readline()

#

le

numero

de

pontos

total

=

int(line)

#

le

os

pontos

e

coloca

na

lista

l

for

i

in

range(total):

line

=

f.readline()

temp

=

line.split()

p

=

Ponto(float(temp[0]), float(temp[1])) l.append(p)

f.close()

return

l

108 / 180

Parte

III

Computação

Científica

com

Python

109 / 180



Workflow

Científico

Gerar

dados

(simulação,
experimentos)

Manipular

e

processar

os

dados

Visualizar

os

resultados

Para

entender,

interpretar

e

validar

o

que

estamos

fazendo

Comunicar

os

resultados

Produzir

figuras

para

relatórios

e

publicações

Apresentações

Objetivo:

apresentar

os

elementos

básicos

da

linguagem

Python

para

escrever

programas

para
solução
computacional
de
problemas
científicos,
manipular,
processar
e
visualizar
os
dados.

110 / 180



- 0
- 0
- 0
- 0
- 0
- 0
- 0
- 0



O
que
é
NumPy?
Numerical
Python
Biblioteca
para
manipulação
de
arrays
multidimensionais
e
matrizes.
Operações
rápidas
em
arrays

(funções
vetorizadas)
Diferença
com
relação
a
listas
tradicionais
do
Python
Vetor
homogêneo
Muito
mais
eficientes
do
que
as
listas
Número
de
elemento
deve
ser

conhecido

a

priori.

O

array

pode

ser

redimensionado

posteriormente.

Muito

eficiente

(implementado

em

C)

111 / 180

Python

Puro

VS

NumPy

#

Python

puro

#

NumPy

import

time

import

time

import

numpy

as

np

l

=

10000000

l

=

10000000

start

=

time.time()

a,

b

=

range(l),

range(l)

start

=

time.time()

c

=

[]

a

=

np.arange(l)

for

i

in

a:

b

=

np.arange(l)

c.append(a[i]

*

b[i])

c

=

a

*

b

t

=

time.time()

-

start

t

=

time.time()

-

start

print("Tempo:

%s"

%

t)

print("Tempo:

%s"

%

t)

Tempo:

4.49

s

Tempo:

0.37



Criando

vetores

NumPy

Arrays

NumPy

podem

ser

criados

a

partir

de

estruturas

de

dados

do

Python

(listas,

tuplas)

ou

a

partir

de

funções

específicas

para

criação

de

arrays.

`zeros((M,N))`

vetor

com

zeros,

M

linhas,

N

colunas

`ones((M,N))`

vetor

com

uns,

M

linhas,

N

colunas

`empty((M,N))`

vetor

vazio,

M

linhas,

N

colunas

`zeros_like(A)`

vetor

com

zeros,

mesmo

formato

de

A

`ones_like(A)`

vetor

com

uns,

mesmo

formato

de

A

`empty_like(A)`

vetor

vazio,

mesmo

formato

de

A

`random.random((M,N))`

vetor

com

numeros

aleatorios,

MxN

`identity(N)`

matriz

identidade

NxN,

ponto

flutuante

`array([[1.5,2,3],[4,5,6]])` cria

a

partir

de

lista

ou

tupla

arange(I,

F,

P)

vetor

com

inicio

I,

fim

F,

passo

P

linspace(I,

F,

N)

vetor

com

N

números

de

I

até

F

113 / 180

Criando

vetores

NumPy

> > >

import

numpy

as

np

> > >

a

=

np.array(

[36.4,

21.6,

15.6,

27.5]

)

#

dtype

=

np.float64

> > >

a

array([

36.4,

21.6,

15.6,

27.5])

> > >

az

=

np.zeros(4)

#

dtype

=

np.float64

> > >

az

array([

0.,

0.,

0.,

0.]


```
> > >
```

```
a
```

```
=
```

```
np.arange(10)
```

```
#
```

```
dtype
```

```
=
```

```
np.int32
```

```
> > >
```

```
a
```

```
array([0,
```

```
1,
```

```
2,
```

```
3,
```

```
4,
```

```
5,
```

```
6,
```

```
7,
```

```
8,
```

```
9])
```

```
> > >
```

```
a
```

```
=
```

```
np.arange(0.0,
```

1.0,

0.2)

#

dtype

=

np.float64

> > >

a

array([

0.

,

0.2,

0.4,

0.6,

0.8])

114 / 180

Criando

vetores

NumPy

> > >

a

=

np.linspace(0.0,

1.0,

6)

> > >

print

a

[

0.

0.2

0.4

0.6

0.8

1.

]

> > >

print

a.size,

a.ndim,

a.shape

#

atributos

importantes

6

1

(6,)

> > >

m

=

a.reshape(2,3)

> > >

print

m

[[

0.

0.2

0.4]

[

0.6

0.8

1.

]]

> > >

print

m.size,

m.ndim,

m.shape

6

2

(2,

3)

> > >

Z

=

np.zeros((3,3))

> > >

print

Z

[[

0.

0.

0.]

[

0.

0.

0.]

[

0.

0.

0.]]

115 / 180

Q



Acessando

arrays

Exemplo

com

array

bidimensional

```
> > >
```

```
a
```

```
=
```

```
np.arange(24)
```

```
> > >
```

```
a
```

```
=
```

```
a.reshape((4,6))
```

```
> > >
```

```
a[2,4]
```

16



Acessando

arrays

```
> > >
```

```
a
```

```
=
```

```
np.arange(24)
```

```
> > >
```

```
a
```

```
=
```

```
a.reshape((4,6))
```

```
> > >
```

```
a[2,4]
```

```
16
```

```
> > >
```

```
a[1]
```

```
array([
```

6,
7,
8,
9,
10,
11])
117 / 180



Acessando

arrays

> > >

a

=

np.arange(24)

> > >

a

=

a.reshape((4,6))

> > >

a[2,4]

16

> > >

a[1]

array([

6,

7,

8,

9,

10,

11])

> > >

a[-1]

array([18,

19,

20,

21,

22,

23])

118 / 180



Acessando

arrays

> > >

a

=

np.arange(24)

> > >

a

=

a.reshape((4,6))

> > >

a[2,4]

16

> > >

a[1]

#

ou

a[1,:]

array([

6,

7,
8,
9,
10,
11])
> > >

a[-1]
array([18,
19,
20,
21,
22,
23])

> > >
a[:,1]

array([
1,
7,
13,
19])

119 / 180





Acessando

arrays

```
> > >
```

```
a
```

```
=
```

```
np.arange(24)
```

```
> > >
```

```
a
```

```
=
```

```
a.reshape((4,6))
```

```
> > >
```

```
a[2,4]
```

```
16
```

```
> > >
```

```
a[1]
```

```
#
```

```
ou
```

```
a[1,:]
```

```
array([
```

```
6,
```

```
7,  
8,  
9,  
10,  
11])  
  
> > >
```

```
a[-1]  
  
array([18,  
19,  
20,  
21,  
22,  
23])
```

```
> > >
```

```
a[:,1]  
  
array([  
1,  
7,  
13,  
19])
```

```
> > >
```

```
a[1:3,:]  
  
array([[  
6,
```

7,
8,
9,
10,
11],
[12,
13,
14,
15,
16,
17]])

120 / 180



Acessando

arrays

> > >

a

=

```
np.arange(24)
```

```
> > >
```

```
a
```

```
=
```

```
a.reshape((4,6))
```

```
> > >
```

```
a[2,4]
```

```
16
```

```
> > >
```

```
a[1]
```

```
#
```

```
ou
```

```
a[1,:]
```

```
array([
```

```
6,
```

```
7,
```

```
8,
```

```
9,
```

```
10,
```

```
11])
```

```
> > >
```

```
a[-1]
```

```
array([18,
```

```
19,
```

20,

21,

22,

23])

> > >

a[:,1]

array([

1,

7,

13,

19])

> > >

a[1:3,:]

array([[

6,

7,

8,

9,

10,

11],

[12,

13,

14,

15,

16,

17]])

> > >

a[1:4,2:5]

array([[

8,

9,

10],

[14,

15,

16],

[20,

21,

22]])

121 / 180



Acessando

arrays

```
> > >
```

```
a
```

```
=
```

```
np.arange(24)
```

```
> > >
```

```
a
```

```
=
```

```
a.reshape((4,6))
```

```
> > >
```

```
a[2,4]
```

```
16
```

```
> > >
```

```
a[1]
```

```
#
```

```
ou
```

```
a[1,:]
```

```
array([
```

```
6,
```

```
7,
```

```
8,
```

```
9,
```

```
10,
```

```
11])
```

```
> > >
```

```
a[-1]
```

```
array([18,
```

```
19,
```

```
20,
```

```
21,
```

```
22,
```

```
23])
```

```
> > >
```

```
a[:,1]
```

```
array([
```

```
1,
```

```
7,
```

```
13,
```

```
19])
```

```
> > >
```

```
a[1:3,:]
```

```
array([[
```

```
6,
```

```
7,
```

```
8,
```

```
9,
```

```
10,
```

```
11],
```

```
[12,
```

13,

14,

15,

16,

17]])

> > >

a[::2,::3]

array([[

0,

3],

[12,

15]])

122 / 180

🔊

Operações

com

arrays

NumPy

suporta

operações

aritméticas

entre

arrays

sem

o

uso

de

loops

com

for

(implementado

em

C)

> > >

import

numpy

as

np

> > >

a,b

=

np.arange(1,11),

np.arange(1,11)

> > >

a

array([

1,

2,

3,

4,

5,

6,

7,

8,

9,

10])

> > >

a

+

1

array([

2,

3,

4,

5,

6,

7,

8,

9,

10,

11])

> > >

a

*

2

array([

2,

4,

6,

8,

10,

12,

14,

16,

18,

20])

> > >

a

*

b

array([

1,

4,

9,
16,
25,
36,
49,
64,
81,
100])
> > >

a
**

3
array([
1,
8,
27,
64,
125,
216,
343,
512,
729,
1000])

Q

Operações

com

arrays

Outras

operações

> > >

```
a=np.array([1,0,1])
```

> > >

```
b=np.array([2,2,4])
```

> > >

```
np.dot(a,b)
```

6

> > >

a

=

```
np.array([1,0,0])
```

> > >

b

=

```
np.array([0,1,0])
```

> > >

```
np.cross(a,b)
```

```
array([0,
```

```
0,
```

```
1])
```

> > >

```
a,b
```

```
=
```

```
np.array([1,2,3]),
```

```
np.array([1,2,3])
```

> > >

```
np.outer(a,b)
```

```
array([[1,
```

```
2,
```

```
3],
```

```
[2,
```

```
4,
```

```
6],
```

```
[3,
```

```
6,
```

```
9]])
```

124 / 180



Funções

e

Arrays

NumPy

Avaliar

funções

usando

arrays

NumPy

from

math

import

exp,

sin

Exemplo:

$f(x) = e \sin(x)$

import

numpy

as

np

Loops

em

vetores

NumPy

def

f(x):

muito

grandes

são

return

exp(sin(x))

lentos

x

=

np.linspace(0.0,

6.0,

100)

Alternativas:

y

=

np.zeros(x.size)

Vectorization

for

i

in

range(x.size): NumPy

oferece

y[i]

=

f(x[i])

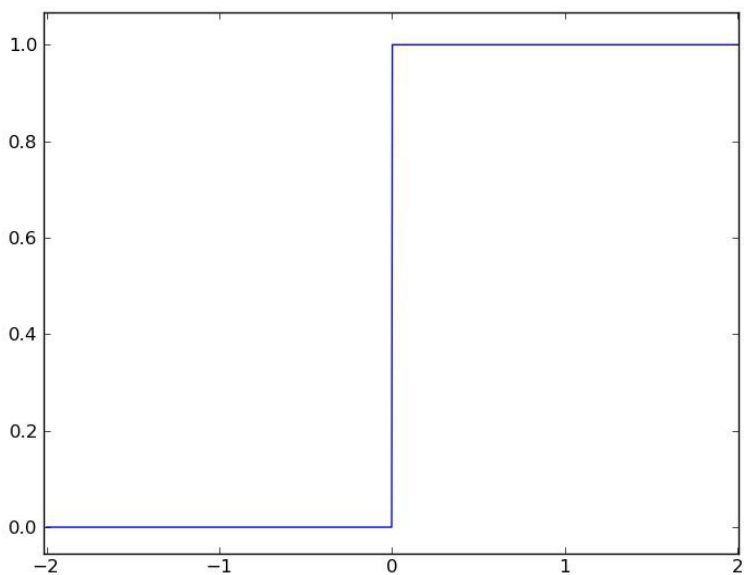
diversas

funções

prontas

125 / 180





Vectorization

Aplicar

f diretamente

em

todo

o

vetor

Muito

mais

eficiente

#

importa

versoes

numpy

de

exp

e

sin

import

numpy

as

np

Mais

compacto

e

fácil

de

ler

def

f(x):

return

np.exp(np.sin(x))

Nem

todas

funções

def

func(x)

estão

prontas

x

=

linspace(0.0,

6.0,

1000)

y

=

f(x)

para

serem

usadas

desta

forma

126 / 180

return

1

else

return

0

if

(x

<

0):

def

H(x):

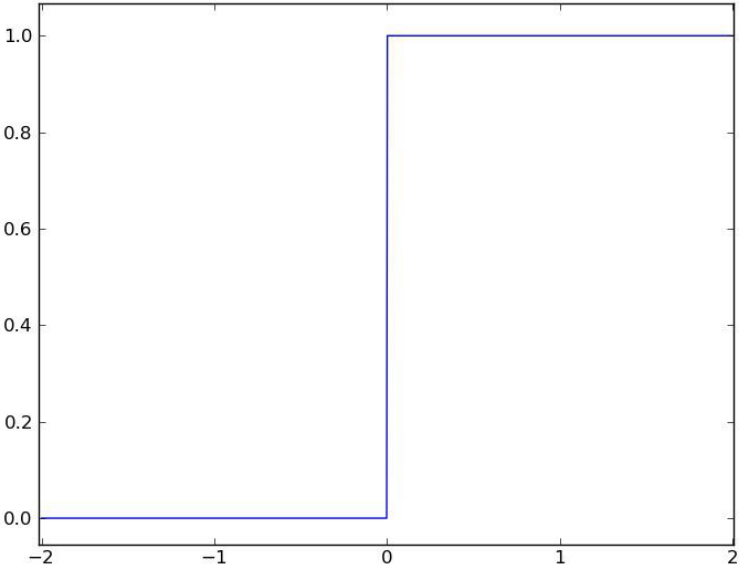
#

funcao

degrau



-
-
-
-



Vectorization

Aplicar

f diretamente

em

todo

o

vetor

Muito

mais

eficiente

#

importa

versoes

numpy

de

exp

e

sin

import

numpy

as

np

Mais

compacto

e

fácil

de

ler

def

f(x):

return

np.exp(np.sin(x))

Nem

todas

funções

def

func(x)

estão

prontas

x

=

linspace(0.0,

6.0,

1000)

y

=

f(x)

para

serem

usadas

desta

forma

#

funcao

degrau

def

H(x):

if

(x

<

0):

return

0

else

return

1

127 / 180





Vectorization

> > >

x

=

linspace(-1,1,5)

array([-1.

,

-0.5,

0.

,

0.5,

1.

])

> > >

x

<

0

array([

True,

True,

False,

False,

False])

Como

vetorizar

funções

assim?

Usar

a

função

where

Uso:

where(condition,

x1,

x2)

Retorna

um

array

do

mesmo

tamanho

de

condition,

onde

o

elemento

i

é

igual

a

x1[i]

se

condition[i]

é

True,

ou

igual

a

x2[i]

caso

contrário

(False).

128 / 180

🔊

🔊

Vectorization

Forma

geral

def

fun_vec(x):

cond

=

<exp_condicao>

x1

=

<expressao1>

x2

=

<expressao2>

return

where(cond,

x1,

x2)

Para

o

exemplo

anterior

temos

def

Hv(x):

cond

=

x

<

0

return

where(cond,

0.0,

1.0)

129 / 180

Alguns

métodos

dos

vetores

a.sum()

soma

todos

elementos

a.min()

menor

elemento

a.max()

maior

elemento

a.mean()

média

aritmética

a.std()

desvio

padrão

a.var()

variância

a.trace()

traço

a.copy()

retorna

cópia

a.conjugate()

complexo

conjugado

> > >

notas

=

np.array([6.,

7.5,

8.,

9.2,

4.3])

> > >

notas.mean()

7.0

> > >

notas. **max()**

9.2

> > >

notas. **min()**

4.3

130 / 180



Copiando

Arrays

A

expressão

a = x faz com que a aponte para o mesmo

array

que

x. Logo, mudanças em a também irão

afetar

x

#

sem

usar

o

metodo

copy()

> > >

x

=

np.array([1.,

2.,

3.5])

> > >

a

=

x

> > >

a[-1]

=

3

#

tambem

altera

x[-1]

> > >

x

array([1.,

2.,

3.])

#

usando

o

metodo

copy()

> > >

x

=

np.array([1.,2.,3.5])

> > >

a

=

x.copy()

> > >

a[-1]

=

9

> > >

a

array([

1.,

2.,

9.]])

> > >

x

array([

1.

,

2.

,

3.5])

131 / 180



Q

Q

Q

Q

Q

Q

•

•

•

•

•

•

Matrizes

Os

arrays

usados

até

então

são

do

tipo

ndarray

NumPy

também

possui

um

tipo

chamado

matrix

Sempre

bidimensional

Algumas

propriedades

especiais

de

matrizes:

matrix.I

(inversa)

matrix.T

(transposta)

matrix.H

(conjugada)

matrix.A

(converte

para

array)

Operador

de

multiplicação

(*) efetua as

operações

usuais

da

Álgebra

Linear

matriz-matriz

matriz-vetor

vetor-matriz

132 / 180

Matrizes

> > >

b

#

vetor

linha

matrix([[2,

1]])

> > >

import

numpy

as

np

> > >

m=np.matrix([[1,

2],

[3,4]])

> > >

b

*

m

#

vet

*

mat

> > >

m

matrix([[5,

8]])

matrix([[1,

2],

[3,

4]])

> > >

b

=

b.T

#

vetor

coluna

> > >

m.I

> > >

b

matrix([[-2.

,

1.

],

array([[2],

[

1.5,

-0.5]])

[1]])

> > >

m.T

matrix([[1,

3],

> > >

m

*

b

#

mat

*

vet

[2,
4]])

matrix([[

4],

[10]])

> > >

b

=

np.array([2,1])

> > >

b

=

np.matrix(b)

> > >

m

*

m.I

#

mat

*

mat

matrix([[1.0000e+00,

1.1102e-16],

[0.0000e + 00,
1.0000e + 00]])

133 / 180



Matrizes

e

Álgebra

Linear

> > >

import

numpy.linalg

as

linalg

Solução

> > >

A

=

np.matrix([[3.,2.,4.],[1.,1.,2.],[4.,3.,-2.]])

> > >

A

de

matrix([[

3.,

2.,

4.],

Sistema

[

1.,

1.,

2.],

[

4.,

3.,

-2.]])

de

Equações

> > >

b

=

np.matrix([[1.],[2.],[3.]])

> > >

b

Lineares

matrix([[

1.],

[
2.],

3x

[
3.]])

$$+ 2 y + 4 z = 1$$

1 x

> > >

x

=

linalg.solve(A,b)

$$+ 1 y + 2 z = 2$$

> > >

x

$$4x + 3y - 2z = 3$$

matrix([[-3.],

[
5.],

[
0.]])

134 / 180



Polinômios

NumPy

define

um

> > >

f

=

np.poly1d([4,2,-1])

> > >

f(3)

tipo

para

41

polinômios

> > >

print

f

Operações

2

aritméticas

e

4

x

+

2

x

-

1

operações

para

> > >

f_1,

F

=

f.deriv(),

f.integ()

derivar,

integrar

> > >

print

f_1

e

avaliar

o

8

x

+

2

polinômio

> > >

print

F

Exemplo:

3

2

$$f(x) = 4x^2 + 2x - 1$$

1.333

x

+

1

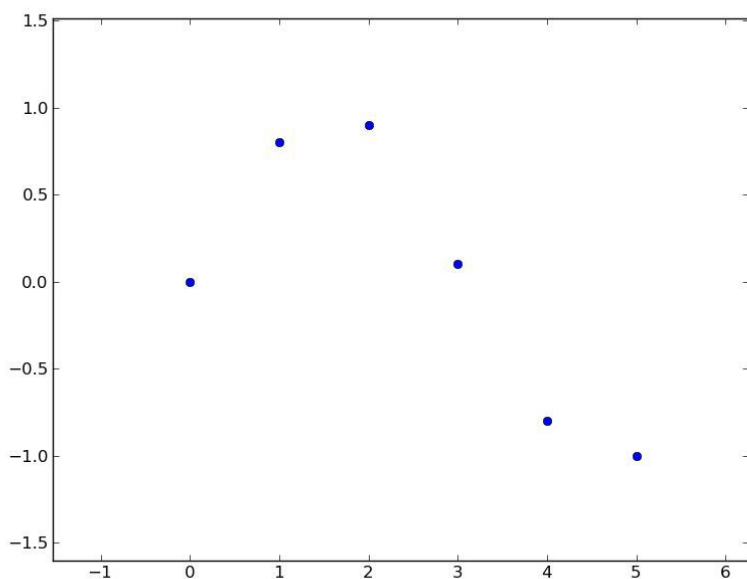
x

-

1

x

135 / 180



Ajuste
de
Curvas
Dado
os
valores
de
uma
função
 $f(x)$ em um

conjunto

de

pontos,

encontrar

uma

função

$g(x)$ que melhor se

aproxime

de

$f(x)$.

Aproximação

polinomial

pelo

método

dos

mínimos

quadrados

$g(x) \Rightarrow$ combinação de

funções

polinomiais

x

0.0

1.0

2.0

3.0

4.0

5.0

`numpy.polyfit(x,y,degree)` $f(x)$

0.0

0.8

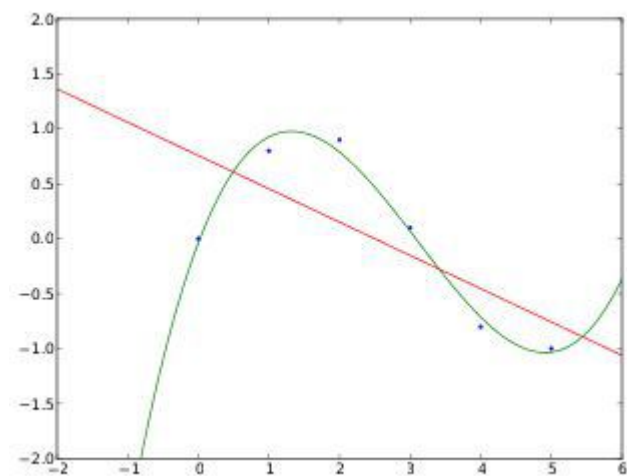
0.9

0.1

-0.8

-1.0

136 / 180



Ajuste

de

Curvas

> > >

import

numpy

as

np

> > >

x=np.array([0.0,

1.0,

2.0,

3.0,

4.0,

5.0])

> > >

y=np.array([0.0,

0.8,

0.9,

0.1,

-0.8,

-1.0])

> > >

c1

=

np.polyfit(x,

y,

1)

> > >

c1

array([-0.30285714,

0.75714286])

> > >

p1

=

np.poly1d(c1)

> > >

c3

=

np.polyfit(x,

y,

3)

> > >

c3

array([

0.08703704,

-0.81349206,

1.69312169,

-0.03968254])

> > >

p1

=

np.poly1d(c3)

137 / 180



SciPy

Coleção

de

algoritmos
matemáticos
e
funções
utilitárias
Implementado
em
cima
do
NumPy
Dividido
em
sub-módulos
constants:
Constantes
físicas
fftpack:
Transformada
Rápida
de
Fourier
integrate:
Integração
numérica

e

ODE

solvers

interpolate:

Interpolação

(Splines)

stats:

Distribuições

e

funções

estatísticas

optimize:

Otimização

sparse:

Matrizes

esparsas

linalg:

Álgebra

Linear

io:

Entrada

e

Saída

signal:

Processamendo

digital

de

sinais

ndimage:

Processamento

digital

de

imagens

138 / 180



❏

❏

Integração

Numérica

com

SciPy

? 4

Exemplo:

$x^2 dx$

0

> > >

from

scipy

import

integrate

> > >

def

fx2(x):

> > >

return

x*x

> > >

integrate.quad(fx2,

0.0,

4.0)

(21.333333333333332,

2.3684757858670003e-13)

> > >

print

4.**3/3

21.3333333333

integrate.quad

usa

um

método

de
quadratura
adaptativa
implementado
em
Fortran
no
pacote
QUADPACK
139 / 180



Integração

Numérica

com

SciPy

Mais

métodos

disponíveis

fixed_quad:

quadratura

Gaussiana

odeint:

integrar

Equações

Diferenciais

Ordinárias

Integrar

um

conjunto

de

dados

discretos

trapz,

simps

e

romb

> > >

x

=

linspace(0.0,

4.0,

25)

> > >

y

=

fx2(x)

array([

0.

,

0.16666667,

0.33333333,

0.5

,

0.66666667,

0.83333333,

1.

,

1.16666667,

1.33333333,

1.5

,

1.66666667,

1.83333333,

2.

,

2.16666667,

2.33333333,

2.5

,

2.66666667,

2.83333333,

3.

,

3.16666667,

3.33333333,

3.5

,

3.66666667,

3.83333333,

4.

])

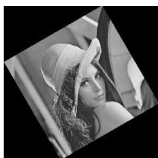
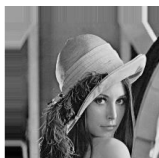
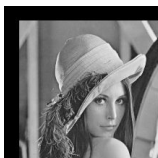
> > >

integrate.trapz(y,

dx = x[1]-x[0])

21.351851851851851

140 / 180



Processamento

Digital

de

Imagens

(ndimage)

> > >

import

scipy

> > >

from

scipy

import

ndimage

> > >

from

pylab

import

*

#

biblioteca

matplotlib

-

proximo

topico!

> > >

lena

=

scipy.lena()

> > >

shifted_lena

=

ndimage.shift(lena,

(50,

50))

> > >

shifted_lena2

=

ndimage.shift(lena,

(50,

50),

mode='nearest')

> > >

rotated_lena

=

ndimage.rotate(lena,

30)

```
> > >
```

```
cropped_lena
```

```
=
```

```
lena[50:-50,
```

```
50:-50]
```

```
> > >
```

```
zoomed_lena
```

```
=
```

```
ndimage.zoom(lena,
```

```
2)
```

```
> > >
```

```
zoomed_lena.shape
```

```
(1024,
```

```
1024)
```

```
> > >
```

```
imshow(lena,
```

```
cmap=cm.gray)
```

```
> > >
```

```
show()
```

```
141 / 180
```



noisy lena



Gaussian filter



median filter



Wiener filter



Processamento

Digital

de

Imagens

> > >

lena

=

scipy.lena()

> > >

import

numpy

as

np

#

cria

uma

versao

com

ruido

> > >

noisy_lena

=

np.copy(lena)

> > >

noisy_lena

+ =

lena.std()*0.5*np.random.standard_normal(lena.shape)

#

aplica

filtros

> > >

blurred_lena

=

ndimage.gaussian_filter(noisy_lena, sigma = 3)

> > >

median_lena

=

ndimage.median_filter(blurred_lena, size = 5)

> > >

import

scipy.signal

#

modulo

de

processamento

digital

de

sinais

> > >

wiener_lena

=

scipy.signal.wiener(blurred_lena, (5,5))

142 / 180



-
-
-
-
-
-
-



Visualização

de

dados

com

matplotlib

A

biblioteca

matplotlib

permite

a

visualização

de

dados

2D

seguindo

o

estilo

do

MATLAB

Gráficos

de

qualidade

para

publicações

Exporta

para

diversos

formatos

Possibilidade

de

embutir

em

interfaces

gráficas

(Qt,

GTK,

...)

Baseado

no

NumPy

e

SciPy

pylab:

módulo

com

diversas

funções

para

plotar

gráficos

de

forma

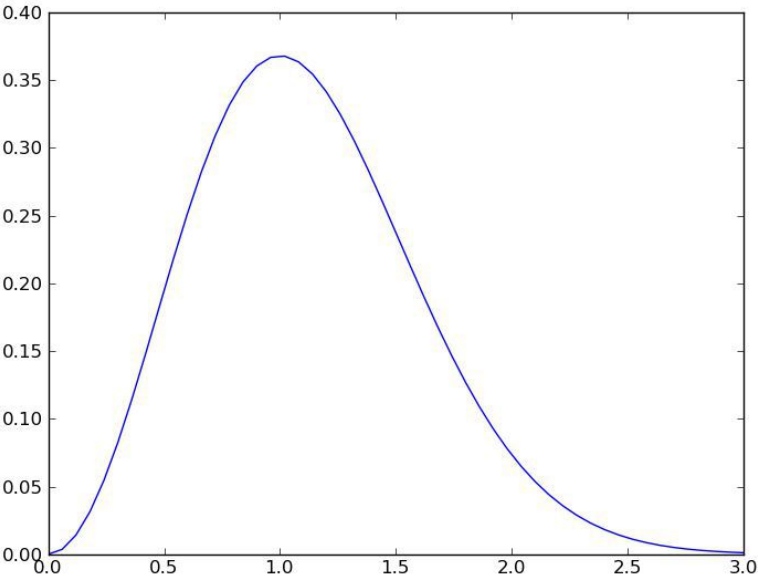
facil

143 / 180



0

0



matplotlib

Exemplo

mais

simples

de

uso:

plot(x,y)

Gráficos

são

gerados

sucessivamente,

i.e.,

cada

chamada

a

função

plot

altera

o

gráfico

> > >

from

pylab

import

*

> > >

x

=

linspace(0,3,51)

> > >

y

=

x**2 * exp(-x**2)

> > >

plot(x,y)

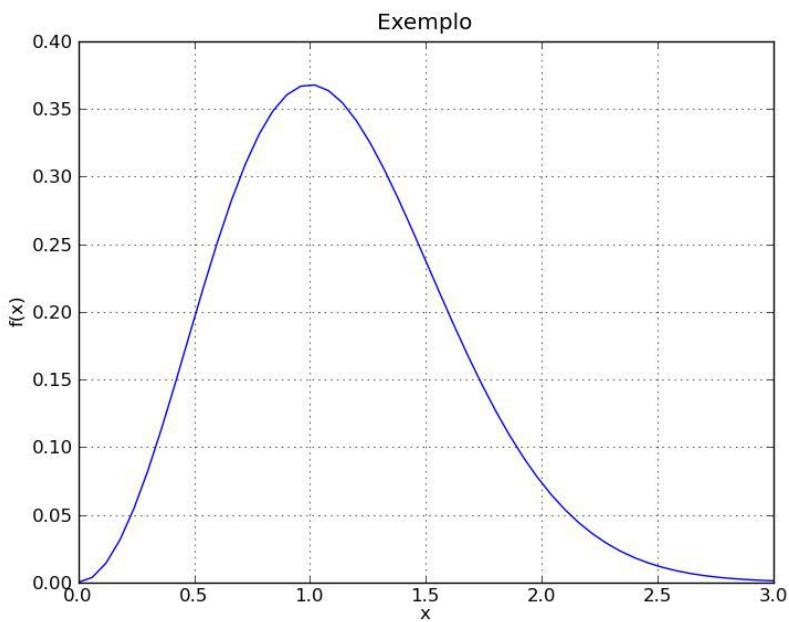
[<matplotlib.lines.Line2D

object...]

> > >

show()

144 / 180



matplotlib

Decorando

o

gráfico

> > >

from

pylab

import

*

> > >

x

=

linspace(0,3,51)

```
> > >
```

```
y
```

```
=
```

```
x**2 * exp(-x**2)
```

```
> > >
```

```
plot(x,y)
```

```
> > >
```

```
grid(True)
```

```
> > >
```

```
xlabel('x')
```

```
> > >
```

```
ylabel('f(x)')
```

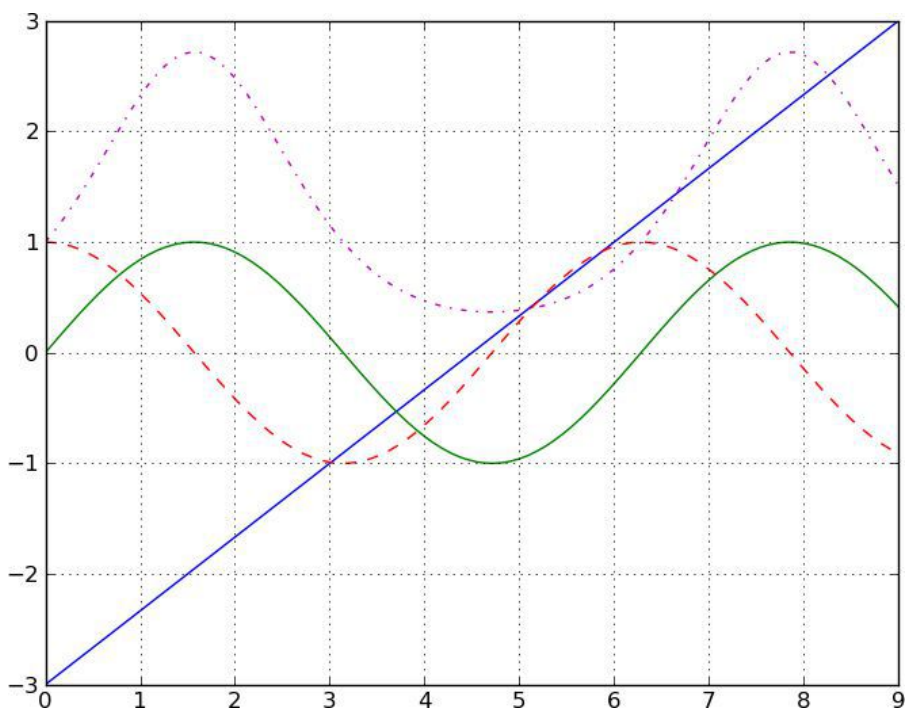
```
> > >
```

```
title("Exemplo")
```

```
> > >
```

```
show()
```

145 / 180



matplotlib

Várias

curvas

> > >

y

=

linspace(-3,

3,

10)

> > >

plot(y)

> > >

x

```
=
```

```
linspace(0,
```

```
9,
```

```
100)
```

```
> > >
```

```
plot(x,sin(x))
```

```
> > >
```

```
plot(x,cos(x),
```

```
'r-
```

```
-.')
```

```
> > >
```

```
plot(x,exp(sin(x)),
```

```
'm
```

```
-.')
```

```
> > >
```

```
grid(True)
```

```
> > >
```

```
show()
```

146 / 180

matplotlib

Controlando

o

estilo

do

plot

A

função

plot

aceita

uma

string

especificando

o

estilo

da

linha

e

do

símbolo

usando

o

seguinte

formato:

'< color > < line > < symbol >'

Cores

r

vermelho

c

ciano

g

verde

m

magenta

b

azul

y

amarelo

w

branco

k

preto

Símbolos

.

pontos

o

circulo

^

triangulo

baixo

s

quadrados

+

cruz

v

triangulo

cima

x

"xis"

*

estrela

<

triangulo

esq

D

diamante

d

diamante

peq.

>

triangulo

dir

Estilo

da

Linha

-

solid

line

-

dashed

line

-,

dash-dot

line

:

dotted

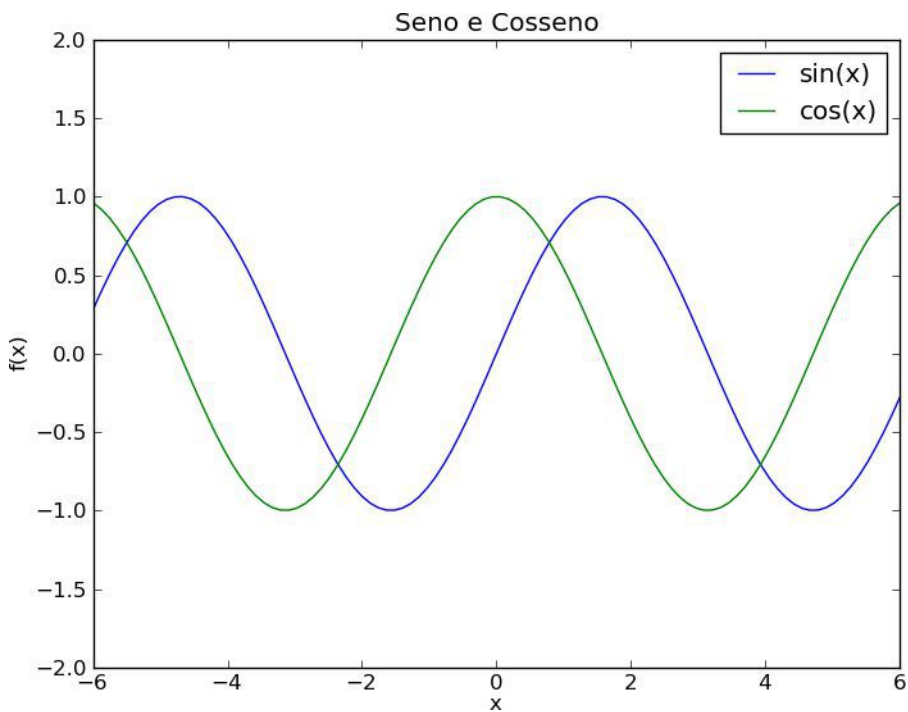
line

147 / 180

☰

☰

☰



matplotlib

Adicionando

mais

informação

ao

gráfico

Legenda

Controle

sobre

os

eixos

> > >

from

```
pylab
```

```
import
```

```
*
```

```
> > >
```

```
x
```

```
=
```

```
linspace(-6,
```

```
6,
```

```
500)
```

```
> > >
```

```
plot(x,sin(x),
```

```
label='sin(x)')
```

```
> > >
```

```
plot(x,cos(x),
```

```
label='cos(x)')
```

```
> > >
```

```
title('Seno
```

```
e
```

```
Cosseno')
```

```
> > >
```

```
xlabel('x')
```

```
> > >
```

```
ylabel('f(x)')
```

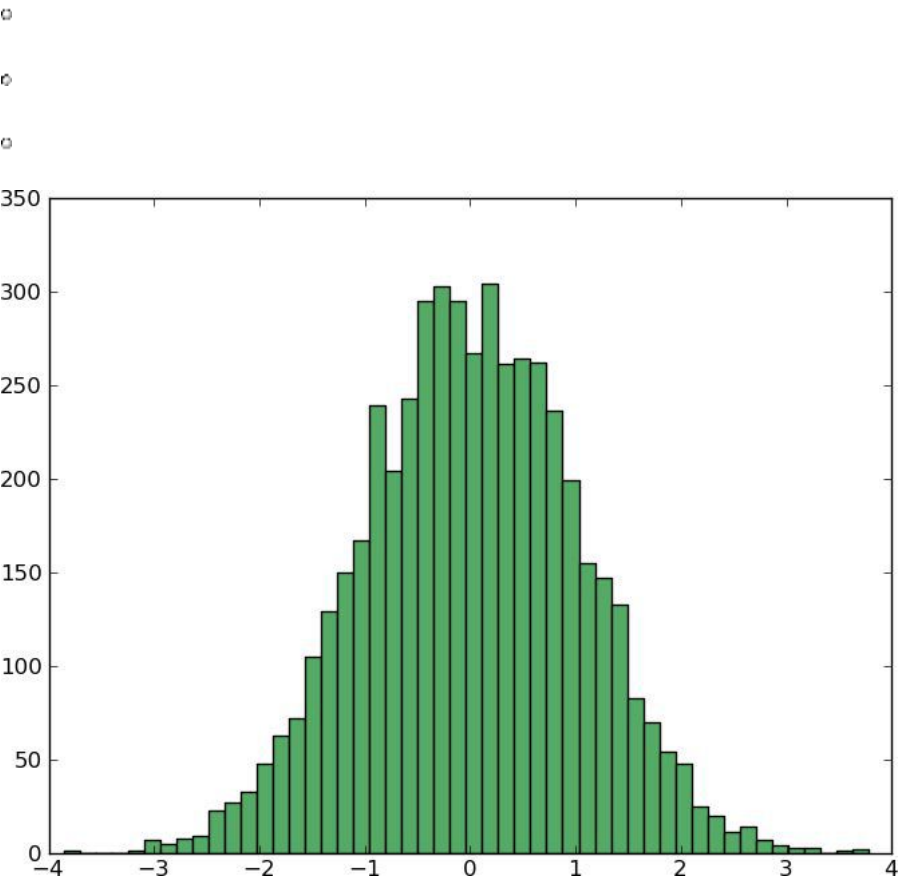
```
> > >
```

axis([-6,6,-2,2])

> > >

legend(loc="upper right")

148 / 180



matplotlib

Histogramas

hist(x,

```
bins = 10)
```

Distribuição

normal

$N(0, 1)$

```
> > >
```

```
from
```

```
pylab
```

```
import
```

```
*
```

```
> > >
```

```
y
```

```
=
```

```
random.randn(1000)
```

```
> > >
```

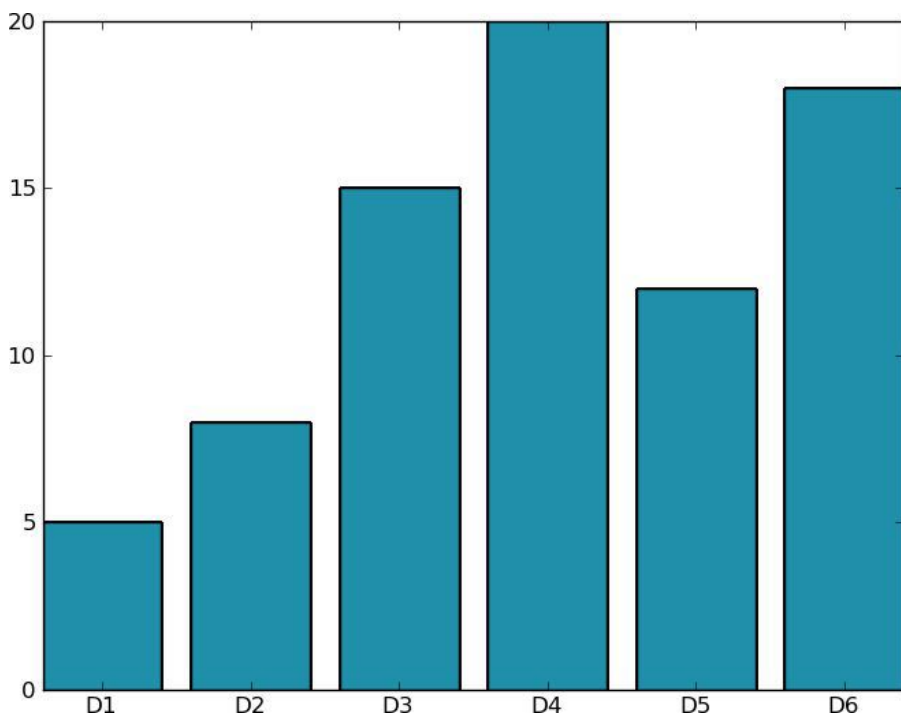
```
hist(y,bins = 50)
```

149 / 180

☺

☺

☺



matplotlib

Gráfico

de

barras

bar(x,

height):

plota

um

gráfico

de

barras

com

retângulos

xticks(x,

labels):

posiciona

rótulos

dos

retângulos

> > >

from

pylab

import

*

> > >

x

=

[1,2,3,4,5,6]

> > >

y

=

[5,8,15,20,12,18]

> > >

bar(x,y,align='center', color='#2090AA')

> > >

lab

=

```
("D1","D2","D3","D4","D5","D6")
```

```
> > >
```

```
xticks(x,
```

```
lab)
```

150 / 180

❏

❏

❏

❏

matplotlib

Salvando

os

gráficos

em

figuras

savefig(filename):

Salva

a

figura

atual

usando

o

formato.

Alguns

parâmetros

opcionais:

format:

'png',

'pdf',

'ps',

'eps',

'svg'

transparent:

True

ou

False

> > >

from

pylab

import

*

> > >

x

=

linspace(-3,3,1000)

> > >

y

=

sin(1/x)

> > >

plot(x,y)

> > >

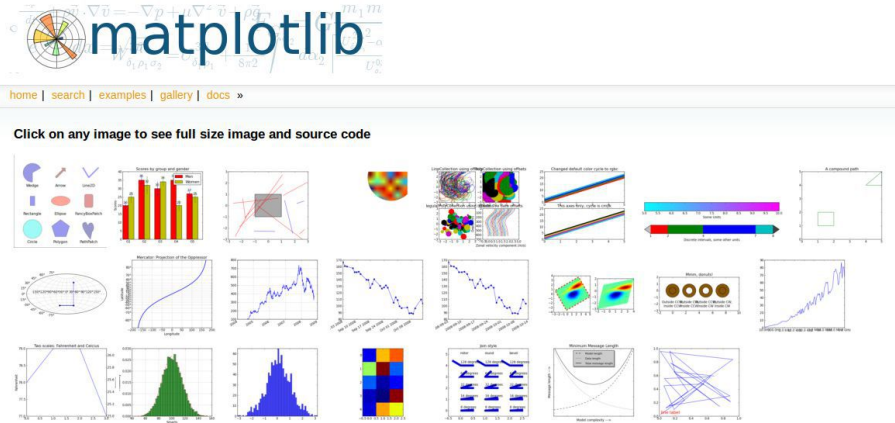
savefig("seno1sx.png")

> > >

savefig("seno1sx.pdf")

> > >

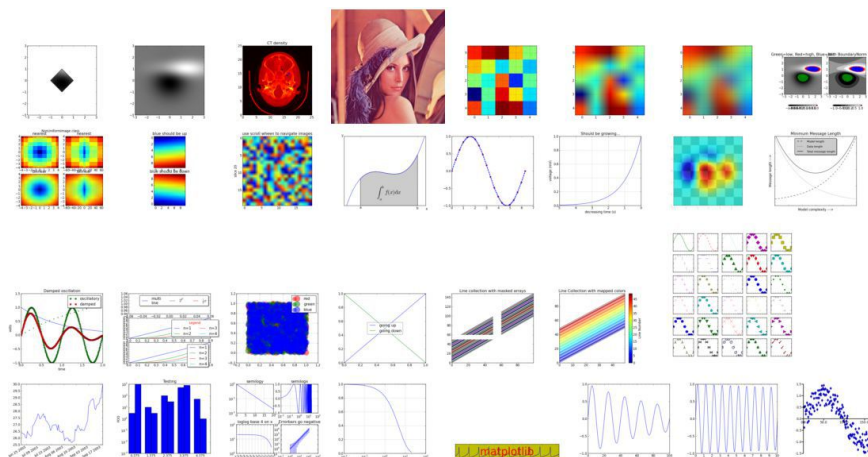
savefig("seno1sx", format="eps") 151 / 180



Galeria

do

152 / 180



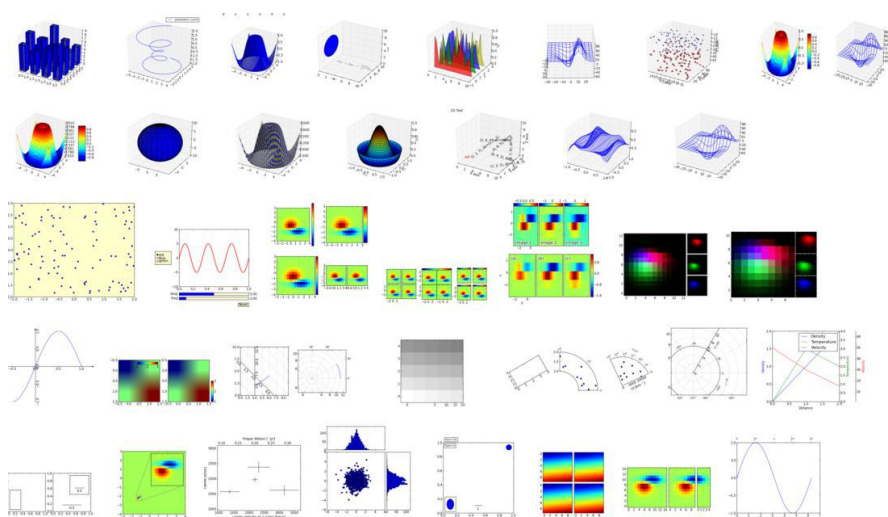
Galeria

do

matplotlib

153 / 180





Galeria

do

matplotlib

matplotlib.sourceforge.net 154 / 180



Exemplo

Completo

Problema:

Resolver

uma

Equação

Diferencial

Ordinária

(EDO)

Encontrar

$u(t)$ tal que

$u'(t) = f(u(t), t)$ dada

a

condição

Inicial

$u(0) = u_0$

Exemplo:

Crescimento

exponencial

(população)

$u'(t) = au$

onde

a é uma constante dada que representa a taxa

de

crescimento

de

u .

0

0

0

0

0

0

Exemplo

Completo

Método

de

Euler

Explícito

$u_{k+1} = u_k + \Delta t f(u_k, t_k)$ onde:

u_k é a aproximação numérica da solução exata $u(t)$ no

tempo

t_k

Δt é o passo de tempo $t_k = k\Delta t$, $k = 0, \dots, n$ Solução

analítica

para

o

exemplo

$u(t) = u_0 \text{eat}$

156 / 180



0

0

0

0

0

Exemplo

Completo

-

Algoritmo

Dado:

$u_0, a, T, \Delta t$

Calcular

n (número de passos de tempo) Para

k

de

0

até

n

faça

Calcular

u_{k+1} usando

$u_{k+1} = u_k + f(u_k, t_k) \Delta t$ Exhibir

os

resultados

157 / 180



Exemplo

Completo

-

Python

$u(t) = u, u_0 = 1, T = 3$

from

pylab

import

*

#

loop

no

tempo

for

k

in

range(n):

#

parametros

u[k + 1]

=

u[k]

+

dt

*

u[k]

u0

=

1

t[k + 1]

=

t[k]

+

dt

T

=

3.0

dt

=

0.1

#

calcula

solucao

exata:

$u(t)$

=

$u0$

$\exp(at)$

n

=

int(T/dt)

v

=

u0

*

$\exp(t)$

#

inicializa

vetores

print

v

u

=

zeros(n + 1)

print

u

v

=

zeros(n + 1)

t

=

zeros(n + 1)

#

exibe

grafico

da

solucao

plot(t,v,

'r-',

label='exata')

#

condicao

inicial

plot(t,u,

'b--',

label='aproximada')

```
t[0]
```

```
=
```

```
0.0
```

```
legend(loc="upper left")
```

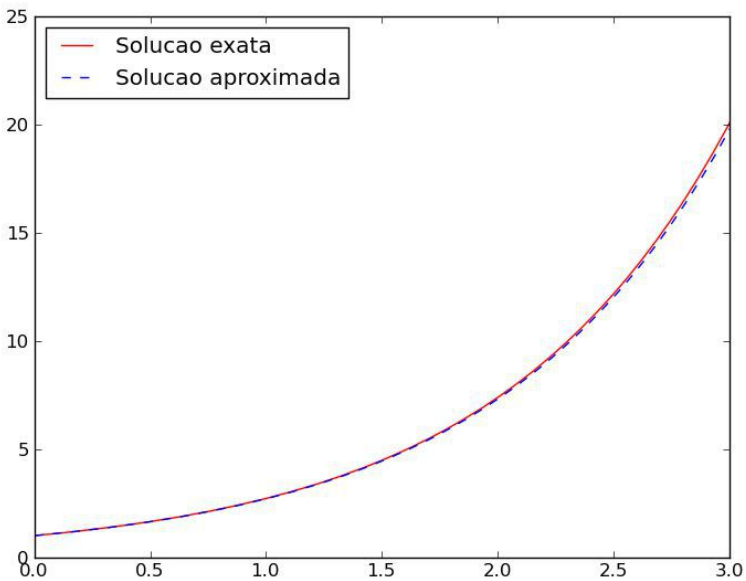
```
u[0]
```

```
=
```

```
u0
```

```
show()
```

```
158 / 180
```



Exemplo

Completo

159 / 180



Exemplo

Completo

com

SciPy

`scipy.integrate.odeint(func, y0,`

`t)`

Usa

a

biblioteca

odepack

escrita

em

FORTRAN.

from

pylab

import

*

from

scipy.integrate

import

odeint

T

=

3.0

u0

=

1.0

dt

=

0.01

n

=

int(T/dt)

def

f(u,t):

return

u

t

=

np.linspace(0.0,

T,

n)

u

=

odeint(f,u0,t)

plot(t,u)

160 / 180

Parte

IV

Outras

bibliotecas

e

projetos

com

Python

161 / 180





Bibliotecas

De

uso

geral

Redes

Games

Interfaces

Gráficas

Tratamento

de

Imagens

Banco

de

Dados

etc,

etc,

etc.

Computação

Científica

Computação

Simbólica

Visualização

Computação

Gráfica

Solução

de

Equações

Diferenciais

Parciais

etc,

etc,

etc.

162 / 180





PyGame

Módulo

multiplataforma

projetado

para

escrever

jogos.

Bibliotecas

de

computação

gráfica

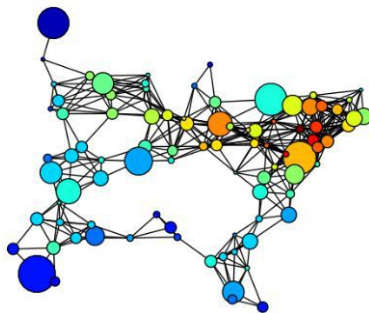
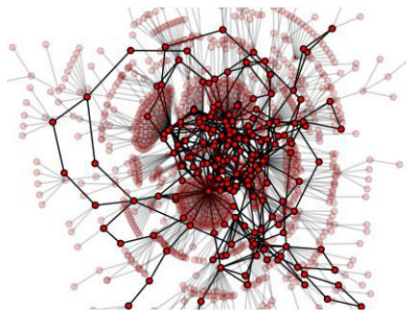
e

som

incluídas.

163 / 180





•

•

•

•

NetworkX

<http://networkx.lanl.gov/>

Estruturas

de

dados

para

grafos,

digrafos

e

multigrafos.

Algoritmos

clássicos

de

grafos.

Adequada

para

operações

em

grandes

grafos.

Pode

ser

usada

para

análise

de

redes.

164 / 180



o

o

Python

Imaging

Library

<http://www.pythonware.com/products/pil> Processamento

de

images.

Suporte

a

diversos

formatos

e

possui

diversas

ferramentas.



**Code less.
Create more.
Deploy everywhere.**

o

o

o

PyQt

<http://www.riverbankcomputing.co.uk/software/pyqt/>

Bindings

Python

para

o

framework

Qt

da

Nokia

Permite

o
desenvolvimento
de
aplicações
com
interface
gráfica
com
o
usuário.
Multiplataforma
166 / 180



Databases

PyGreSQL

(PostgreSQL):

<http://www.pygresql.org/>

psycopg

(PostgreSQL):

<http://initd.org/psycpg/>

sqlite3

(SQLite):

<http://docs.python.org/library/sqlite3.html> import

sqlite3

conn

=

sqlite3.connect('example.db') c

=

conn.cursor()

#

Create

table

c.execute(*"""CREATE*

TABLE

stocks

(date

text,

trans

text,

symbol

text,

qty

real,

price

real)''')

#

Insert

a

row

of

data

c.execute("INSERT

INTO

stocks

VALUES

('2006-01-05','BUY','RHAT',10,5.14)')

#

Save

(commit)

the

changes

conn.commit()

#

We

can

also

close

the

cursor

if

we

are

done

with

it

c.close()

167 / 180

🔍

🔍

🔍

🔍

🔍

🔍

Mechanize

<http://wwwsearch.sourceforge.net/mechanize/>

Web

browsing

com

Python

Permite:

Abrir

qualquer

URL

Preencher

formulários

HTML

Visualizar

histórico

etc,

etc,

etc

import

re

import

mechanize

br

=

mechanize.Browser()

br. **open**("http://www.example.com/")

#

follow

second

link

with

element

text

matching

regular

expression

response1

=

br.follow_link(text_regex=r"cheese\s*shop", nr=1) **assert**

br.viewing_html()

print

br.title()

print

response1.geturl()

print

response1.info()

#

headers

print

response1.read()

#

body

Bibliotecas

para

Computação

Científica

169 / 180



▼

◊

◊

Sympy

Computação

Simbólica

Alternativa

livre

aos

softwares

Maple,

Mathematica

e

Matlab.

Aritmética

básica,

expansões,

funções,

derivadas,

integrais,

substituições,

limite,

matrizes,

etc.

> > >

from

sympy

import

*

> > >

x

=

Symbol("x")

> > >

x

=

Symbol('x')

> > >

limit(sin(x)/x,

x,

0)

> > >

f

=

2

*

cos(x)

1

> > >

diff(f,

x)

> > >

limit(1/x,

x,

oo)

-2*sin(x)

0

170 / 180



0

0

0

0

Sage

Software

matemático

livre

com

licença

GPL.

Alternativa

livre

aos

softwares

Maple,

Mathematica

e

Matlab.

Re-utiliza

pacotes

como

Maxima,

GAP,

Pari/GP,

softwares

de

renderização

de

imagens

e

outros.

Disponível

para

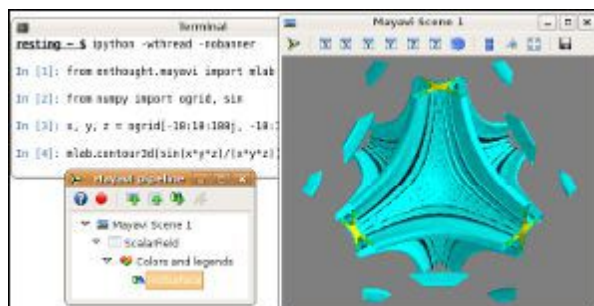
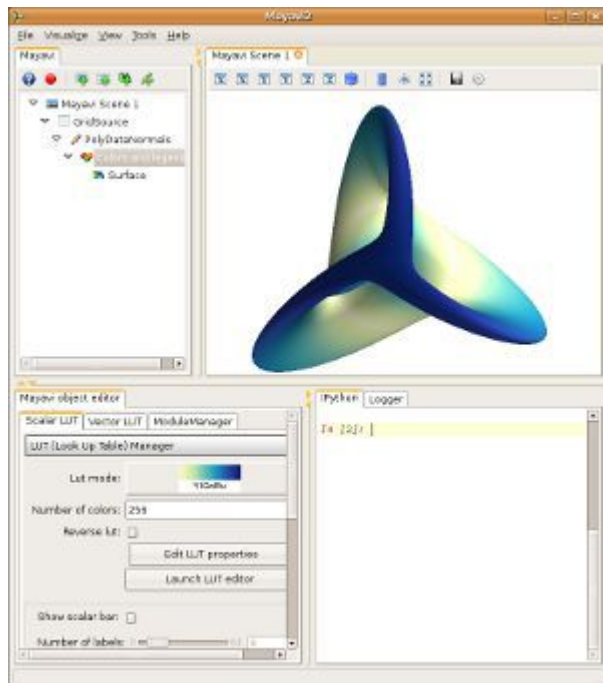
uso

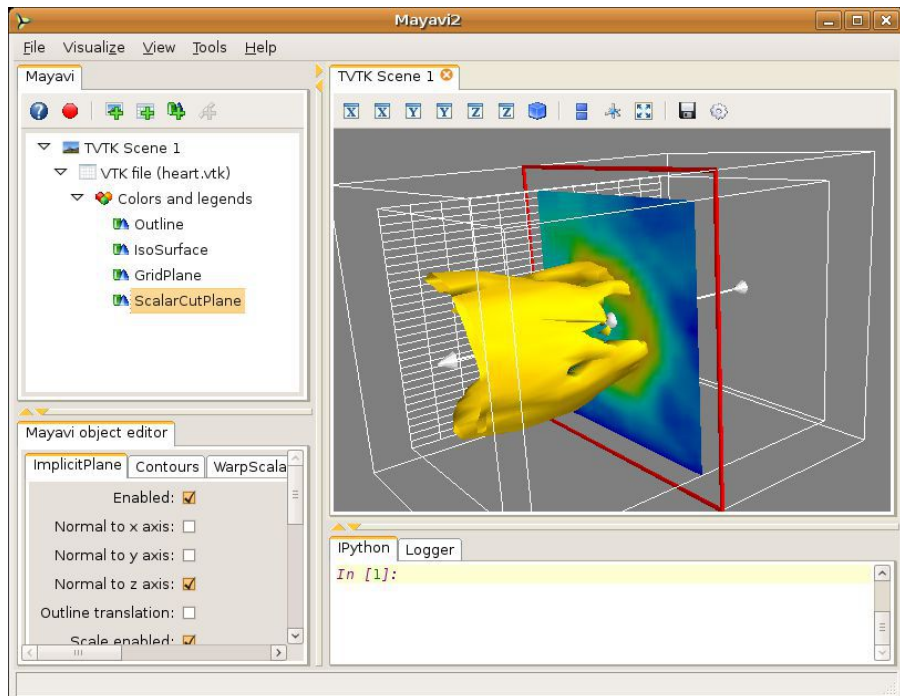
online

via

browser.

171 / 180





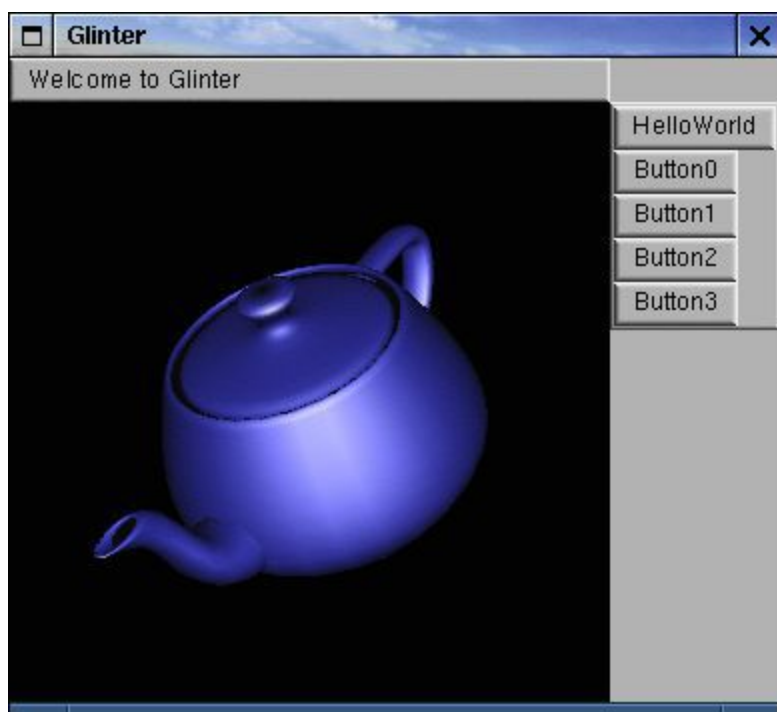
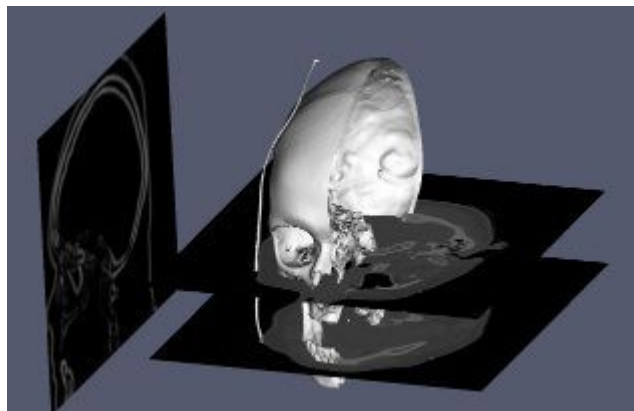
Visualização

Científica

MayaVi

172 / 180





Computação

Gráfica,

Visualização

Computação

gráfica,

processamento

de

imagens

e

visualização.

Escrito

em

C + +

com

interface

em

Tcl/Tk,

Java

e

PyOpenGL:

binding

Python.

de

OpenGL

para

Python

OpenGL:

API

livre

utilizada

na

computação

gráfica

173 / 180



Álgebra

Linear

Computacional

Algebraic

Multigrid

Solvers

PySparse:

in

Python

Biblioteca

para

matrizes

esparsas

Diversas

implementações

do

AMG

Diversos

formatos.

Conversão

Fácil

de

usar,

rápido,

eficiente

Solvers

iterativos

Precondicionadores

<http://code.google.com/p/pyamg/>

<http://pysparse.sourceforge.net/>

174 / 180



FENICS
PROJECT

9

2

9

9



2

3

3

2

Solução

Numérica

de

Equações

Diferenciais

FEniCS

Project

Solução

automatizada

de

EDPs

usando

o

FiPy

(A

finite

volume

método

dos

elementos

PDE

solver

written

in

finitos

Python)

Alto

nível

de

Solver

de

EDPs

usando

abstração

(Muito

o

método

dos

volumes

próximo
da
formulação
finitos
matemática)
Orientado
a
objetos
Paralelismo,
Computação
paralela
adaptatividade,
estimativas
de
erro.

175 / 180



将

sho

軍

gun

0

0



0

0

0

Apredizagem

de

Máquina

Scikits

Construído

sobre

Shogun:

A

Large

Scale

NumPy,

SciPy

e

Machine

Learning

Matplotlib

Toolbox

Diversas

técnicas

como

SVM

(Support

Vector

SVM,

K-Means,

etc

Machines)

<http://www.shogun-toolbox.org/>



-
-
-
-
-
-





Python

para

Física

Astrophysics:

<http://packages.python.org/Astrophysics/>

Utilitários

de

astrofísica

em

Python

PyFITS:

<http://packages.python.org/pyfits/>

Manipulação

de

imagens

FITS

em

Python

YT:

<http://yt-project.org/>

yt

é

um

toolkit

para

manipular

dados

de

simulações

astrofísicas

com

suporte

para

análise

e

visualização.

177 / 180

🔊

🔊

🔊



Python

para

Química

Cheminformatics:

OpenBabel

(Pybel),

RDKit,

OEChem,

Daylight

(PyDaylight),

Cambios

Molecular

Toolkit,

Frowns,

PyBabel

and

MolKit

Computational

chemistry:

OpenBabel,

cclib,
QMForge,
GaussSum,
PyQuante,
NWChem,
Maestro/Jaguar,
MMTK

Visualisation:

CCP1GUI,
PyMOL,
PMV,
Zeobuilder,
Chimera,
VMD

178 / 180



The
Zen
of
Python
Beautiful
is

better

than

ugly.

Explicit

is

better

than

implicit.

Simple

is

better

than

complex.

Complex

is

better

than

complicated.

Flat

is

better

than

nested.

Sparse

is
better
than
dense.
Readability
counts.
Special
cases
aren't
special
enough
to
break
the
rules.
Although
practicality
beats
purity.
Errors
should
never
pass
silently.

Unless
explicitly
silenced.

In
the
face
of
ambiguity,
refuse
the
temptation
to
guess.

There
should
be
one-
and
preferably
only
one
-obvious
way
to

do

it.

Although

that

way

may

not

be

obvious

at

first

unless

you're

Dutch.

Now

is

better

than

never.

Although

never

is

often

better

than

right

now.

If

the

implementation

is

hard

to

explain,

it's

a

bad

idea.

If

the

implementation

is

easy

to

explain,

it

may

be

a

good

idea.

Namespaces

are

one

honking

great

idea

-

let's

do

more

of

those!

179 / 180

```
from numpy import *
from ODESolver import RungeKutta4
```

```
def rhs(u, t):
    R = 1
    return alpha*u*(1 - u/R)
```

$$\frac{du}{dt} = \alpha u(1 - u/R)$$

$$u(0) = 0.1$$

$$R = 1$$

$$\alpha = 0.2$$

TEXTS IN COMPUTATIONAL SCIENCE
AND ENGINEERING

6

Hans Petter Langtangen

A Primer on Scientific Programming with Python

Editorial Board

T. J. Barth

M. Griebel

D. E. Keyes

R. M. Nieminen

D. Roose

T. Schlick

 Springer

Referências

e

Agradecimentos

Hans

Petter

Langtangen

-

"A

Primer

on

Scientific

Computing

with

Python"

Slides

de

Rafael

Sachetto

Oliveira

(UFSJ)

Mais

informações:

<http://www.python.org> <http://numpy.org>

<http://fperez.org/py4science/>

Equipe

da

Semana

da

Computação

(USP

-

São

Carlos)

180 / 180